Algorithmen für Chaos und Fraktale

Dietmar
Herrmann







ADDISON-WESLEY



Dietmar Herrmann

Algorithmen für Chaos und Fraktale



ADDISON-WESLEY PUBLISHING COMPANY

Bonn · Paris · Reading, Massachusetts · Menlo Park, California · New York
Don Mills, Ontario · Wokingham, England · Amsterdam · Milan · Sydney
Tokyo · Singapore · Madrid · San Juan · Seoul · Mexico City · Taipei, Taiwan

Die Deutsche Bibliothek – CIP-Einheitsaufnahme

Herrmann, Dietmar
Algorithmen für Fraktale und Chaostheorie / Dietmar Herrmann. – Bonn; Paris;
Reading, Mass. [u. a.]: Addison-Wesley, 1994
ISBN 3-89319-633-1





1. Auflage 1994

Satz: Reemers EDV-Satz, Krefeld. Gesetzt aus der Sabon 10/12 Pkt.
Belichtung, Druck und Bindung: Paderborner Druck Centrum, Paderborn
Produktion: Claudia Lucht, Bonn
Umschlaggestaltung: Hommer Grafik-Design, Haar bei München

Das verwendete Papier ist aus chlorfrei gebleichten Rohstoffen hergestellt und alterungsbeständig. Die Produktion erfolgt mit Hilfe umweltschonender Technologien und unter strengsten Auflagen in einem geschlossenen Wasserkreislauf unter Wiederverwertung unbedruckter, zurückgeführter Papiere.

Inhaltsverzeichnis

Vorwort

Das Buch der Natur ist mit mathematischen Symbolen geschrieben.
G. Galileo

Die Chaostheorie und fraktale Geometrie hat inzwischen ihren Siegeszug um die Welt angetreten. Kaum eine Zeitschrift versäumte es, ihre Leser mit wundersamen und kunstvollen Bildern an der Entdeckungsreise in das Reich des Apfelmännchens und der Julia-Mengen teilhaben zu lassen.

Zwischen den Anforderungen, die eine populäre, auf Anschauung gerichtete Einführung und eine rein wissenschaftlich orientierte Abhandlung stellt, klafft eine große Lücke. Hier setzt das vorliegende Buch ein. Es liefert die mathematischen Grundlagen für die Theorie der dynamischen Systeme auf einem mittleren Niveau. Dabei werden nur elementare Kenntnisse der Differential- und Integralrechnung, der analytischen Geometrie und der komplexen Zahlen vorausgesetzt. Es vermittelt fundierte Einblikke in folgende Bereiche der dynamischen Systemtheorie:

- Fixpunkt-Theorie
- Ljapunow-Exponenten
- Bifurkationen nach Feigenbaum, Hopf
- Phasen-Diagramme
- Attraktoren
- Poincaré-Schnitte
- Dimensionsbegriffe
- KAM-Theorie

Computerprogramme aus allen gängigen Programmiersprachen liefern mathematische Algorithmen für

- Iterierte Funktionssysteme
- Lindenmayer-Systeme
- Julia-Mengen
- Mandelbrot-Mengen
- Fraktale Kurven
- Zellular-Automaten

Besonderern Wert wurde auf die Darstellung von Anwendungen gelegt. Der Leser erfährt wichtige Auswirkungen des Chaos in der Natur, z.B. aus den Bereichen

- Chemie (oszillierende Reaktionen)
- Physik (nichtlineares Pendel, Isospin-Magnetismus)
- Elektronik (Diodenschaltkreis, Oszillator)
- Astronomie (Saturnringe)
- Populationsdynamik (Volterra-Gleichung, Wirt-Parasit-Modelle)
- Clusterbildung

Ein ausführliches Einleitungskapitel führt dem Leser die Entstehung der Chaostheorie und der fraktalen Geometrie vor Augen. Die wichtigsten Begriffe der dynamischen Systemtheorie werden in Kapitel 19 in Stichwort-Form erläutert. Circa 200 Abbildungen dienen dazu, die Anschaulichkeit zu erhöhen.

Neben der Vielzahl der im Buch aufgelisteten Programme enthält die beigefügte Diskette zahlreiche weitere Programme, die aus Umfangsgründen nicht abgedruckt wurden. Die Programme sind in jeweils einem Unterverzeichnis der Diskette zusammengefaßt, das wie die Kapitelnummer lautet. Die Programme tragen den Namen, der im Fließtext oder in der Übungsaufgabe genannt ist.

Das schon erwähnte Kapitel 1 bietet einen Einblick in die Entstehungsgeschichte der Chaostheorie, die nicht von Mathematikern initiiert wurde, sondern durch Forschungsbeiträge aus verschiedenartigen Gebieten wie Wetterkunde, Medizin, Chemie, Astronomie und Biologie entstand.

Kapitel 2 behandelt die eindimensionalen dynamischen Systeme exemplarisch am Beispiel der logistischen Gleichung. Zweidimensionale Systeme werden in den Kapiteln 3 bis 5 besprochen. Kapitel 3 zeigt die Kreisgleichung auf, Kapitel 4 die gekoppelte logistische Gleichung und Kapitel 5 das Hénon-System. In Kapitel 6 findet sich eine ausführliche Diskussion des dreidimensionalen Lorenz-Systems.

Kapitel 7 vermittelt einen Eindruck über die weiteren bekannten Attraktoren, z.B. von Rössler, Ueda, Duffing und anderen. Konservative Systeme und Anwendungen der KAM-Theorie werden im Kapitel 8 anhand der quadratische Hénon-Gleichung, der Twist-Map und des Hénon-Heiles-Systems besprochen. Den Weg ins Chaos liefern die iterativen Systeme wie die Hufeisen-, Katzen- und Bäcker-Abbildung des Kapitels 9.

Das Chaosspiel von Kapitel 10 führt in die fraktale Geometrie ein. Die wichtigsten fraktalen und selbstähnlichen Kurven von Koch, Sierpinski, Lévy, Cesàro, Peano und anderen finden sich in den Kapiteln 11 und 12, ebenso die Bäume nach Cayley und Pythagoras.

Das Kapitel 13 ist den Iterierten Funktionssystemen (IFS) nach Barnsley gewidmet. Hier findet der Leser den IFS-Code für fraktale Kurven und Pflanzenobjekte wie Far-

ne, Blätter und Bäume. Kapitel 14 ergänzt diese Darstellung durch die Turtle-Graphik und liefert damit die Theorie der Lindenmayer-Systeme.

Kapitel 15 zeigt grundlegende Eigenschaften der Julia-Mengen auf und liefert vielfältige Algorithmen zur Erzeugung von Julia-Graphiken, darunter das Distanz-Verfahren, die Backtracking-Methode und die inverse Iteration. Eng damit zusammen hängt die Theorie der Mandelbrot-Menge, die im folgenden Kapitel 16 geschildert wird. Die Darstellung beschränkt sich aber nicht nur auf die Mandelbrot-Menge der quadratischen Funktion, sondern umfaßt auch die trigonometrischen und Exponentialfunktionen.

Den Anwendungen gewidmet sind die umfangreicheren Kapitel 17 und 18. Das erstgenannte enthält Beispiele aus Chemie, Elektronik und Physik. Das folgende Kapitel behandelt Biologie, Populationsdynamik, Zellularautomaten und Clusterbildung (DLA).

Dem Verlag danke ich für die Herausgabe und gute Ausstattung dieses Buchs mit Farbtafeln und Diskette, ebenso für die Hilfe bei der Beschaffung von amerikanischer Literatur. Meinem Kollegen Herrn Prof. Dr. Dietrich Schwägerl von der FH München (Fachbereich Mathematik/Informatik) verdanke ich manche Anregungen. Wesentliche Unterstützung bei der Literatursuche verdanke ich Herrn StR Franz Moser (Haag). Zum besonderen Dank bin ich Herrn Dr. Leo Klingen (Bonn) verpflichtet, der mir wertvolle Hinweise für mein Manuskript gab.

Allen Lesern dieses Buchs wünsche ich eine anregende Lektüre!

Dietmar Herrmann, März 94

Hinweise zu den Graphik-Programmen

Die im diesem Buch enthaltenen Programme sind für den schon längst zum Standard gewordenen VGA-Modus geschrieben. Sie können aber mit nur geringen Änderungen für andere Auflösungen umgeschrieben werden.

Mit der `screen`-Anweisung kann in Quick Basic der Graphik-Modus gewählt werden. Das bei MS-DOS ab Version 5.0 mitgelieferte Q-Basic ist fast vollständig kompatibel zum Quick Basic-Compiler, ist aber als Interpretersystem wesentlich langsamer. Weitgehend kompatibel ist auch das frühere Turbo Basic, das nun Power Basic heißt. Folgende Parameterwerte sind in der `screen`-Anweisung erlaubt

- `screen 0` Textmodus
- `screen 1` CGA-Modus 320×200 Punkte
- `screen 2` CGA-Modus 640×200 Punkte

- `screen 7` EGA-, VGA-Modus 320×200 Punkte
- `screen 8` EGA-, VGA-Modus 640×200 Punkte
- `screen 9` EGA-, VGA-Modus 640×350 (64K-Videospeicher)
- `screen 10` EGA-, VGA-Modus 640×350 Punkte
- `screen 11` VGA-, MCGA-Modus 640×480 Punkte (2 Farben)
- `screen 12` VGA-Modus 640×480 Punkte (16 Farben)
- `screen 13` VGA-, MCGA-Modus 320×200 Punkte (256 Farben)

Besitzer einer Hercules-Karte müssen zuerst das Programm `qbherc.com` oder `msherc.exe` starten. In diesem Modus gibt es nur eine Farbe. Der Bildschirmausschnitt in Quick Basic kann mit dem `window`-Befehl unabhängig von der Bildschirmauflösung gewählt werden. Beispielsweise definiert `window(-2,-3)-(4,5)` einen Bildschirmauschnitt im Bereich $\{-2 \le x \le 4\} \times \{-3 \le y \le 5\}$.

In Turbo Pascal gibt es den `detect`-Befehl, der die installierte Graphikkarte des Rechners weitgehend automatisch bestimmt. Der zugehörige Programmabschnitt lautet

```
Graphdriver := detect;
InitGraph(Graphdriver,GraphMode,'\tp\bgi');
SetGraphMOde(Graphmode);
```

Dabei müssen sich die BGI (*Borland Graphics Inferface*)-Dateien, z.B. `egavga.bgi` in dem angegebenenen Verzeichnis befinden. Hat man eine VGA-Karte, so kann der VGA-Modus direkt programmiert werden mittels

```
GraphDriver = 4; Graphmode := VGAHi;
InitGraph(Graphdriver,GraphMode,'\tp\bgi');
```

Bei Benutzung der Graphik-Befehle muß zuvor die Graphik-Bibliothek mit dem Befehl `uses graph` eingebunden werden.

Die neueren Graphikkarten verfügen in der Regel über einen Super-VGA-, VESA- oder IBM8514-Modus. Dazu muß vor dem Aufruf der BGI-Datei, z.B. `svga256.bgi` oder `ibm8514.bgi` der entsprechende Treiber von der Hersteller-Diskette geladen werden. Bei Hercules-Karten muß vor dem Aktivieren von `herc.bgi` ebenfalls das entsprechende Programm aufgerufen werden. Für VESA-Karten befindet sich bei dem Dateisystem von Turbo Pascal 7.0 ein separates Programm, das in den VESA-Graphikmodus umschaltet.

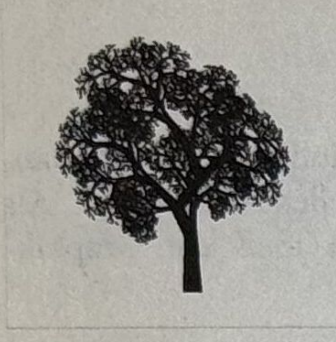
Turbo C benützt ebenfalls das BGI-Interface. Daher gilt das für Turbo Pascal Gesagte hier analog. Die Graphik kann gestartet werden mit dem Programmausschnitt

```
gdriver = DETECT;
initgraph(&gdriver,&gmode,"\\tc\\bgi");
```

Der VGA-Modus wird initialisiert mit

```
gdriver = 4; gmode = VGAHI;
initgraph(&gdriver,&gmode,"\\tc\\bgi");
```

Auch hier muß sich der BGI-Treiber im angegebenen Pfad befinden. Ist die Pfadangabe die leere Zeichenkette, so sucht der Compiler im aktuellen Verzeichnis. Als Header-Datei wird stets `graph.h` benötigt. Beim Compilieren muß die Graphik-Bibliothek `graphics.lib` eingebunden werden.



1 Einleitung: Wie alles anfing

Von demjenigen, der die Geschichte irgendeines Wissens überliefern will, können wir mit Recht verlangen, daß er uns Nachricht gebe, wie die Phänomene nach und nach bekannt geworden und was man darüber phantasiert, gewähnt, gemeint und gedacht habe.
J. W. von Goethe

Chaos ist heutzutage ein Modewort geworden und wird meist als Superlativ von Unordnung verwendet. Das Wort »Chaos« leitet sich aus dem griechischen Wort χαος her und bedeutet ursprünglich den unendlich leeren Raum oder die gestaltlose Urmasse. Ετοι μεν πρωτιστα χαος γενετ´ - *Wahrlich zuerst entstand das Chaos*, schreibt Hesiod in seiner Theogonie. Im Lateinischen wurde das Wort in ähnlicher Bedeutung übernommen.

Ante mare et terras et quod tegit omnia caelum unus erat toto naturae vultus in orbe, quem dixere chaos...

schreibt Ovid in seinen Metamorphosen. Das Chaos ist damit die Urform der Erde, bevor sie von Gott mit Ordnung und Harmonie gefüllt wird; ganz im Sinne des *Tohu-wa-bohu* aus der Genesis (1. Buch Moses): *Und die Erde war wüst und leer und es ward finster über der Tyiefe*. Chaos im heutigen Sprachgebrauch bedeutet den Zustand der Unordnung und Irregularität.

Die Philosophie hat sich schon lange mit dem Begriff des Chaos auseinandergesetzt. Die Philosophen sind daher skeptisch über die Einvernahme des Chaos durch die moderne Naturwissenschaft. *Die Erfahrung des Chaos unterliegt in der Geschichte des Denkens einer Logik der Verdrängung und Wiederkehr*, schreibt N. Bolz von der FU Berlin in seinem Buch »Die Welt als Chaos und als Simulation« (Fink Verlag 1992). Kennzeichnend stellt er seinem Vorwort ein Zitat von L. Wittgenstein voran:

Beim Philosophieren muß man in´s alte Chaos hinabsteigen, und sich dort wohlfühlen. Steigen sie immer von den kahlen Höhen der Gescheitheit in die grünenden Täler der Dummheit.

In der Physik versteht man chaotische Vorgänge als zeitlich und räumlich nicht vorhersagbare Bewegungen in einem Phasenraum. Unter *deterministischem* Chaos versteht man das chaotische Verhalten eines Systems, dessen Anfangsbedingungen und Wirkungsgesetz vollständig bekannt sind. Poincaré entdeckte 1892, daß ein mechanisches System mit einer wohldefinierten Hamilton-Funktion durchaus eine chaotische Bewegung zeigen kann. Bis dahin hatten die Physiker angenommen (und in vielen Physikbüchern liest man dies noch implizit), daß alle eindeutig formulierten Probleme in Form einer Differential- oder Differenzengleichung gutartige und reguläre Lösungen aufweisen würden.

In der amerikanischen Literatur wird die Entdeckung der Chaostheorie als dritte wissenschaftliche Revolution in diesem Jahrhundert nach der Relativitätstheorie und Quantenmechanik gefeiert, im Sinne des Wissenschaftstheoretikers Thomas S. Kuhn:

Die Relativitätstheorie Einsteins beendete die Newtonsche Illusion von Zeit und Raum als absolute Kategorien, die Quantentheorie setzte dem Newtonschen Traum von einem exakt kontrollierbaren Meßprozeß ein Ende, und nun erledigt die Chaostheorie die Laplacesche Utopie von der deterministischen Vorhersagbarkeit.

schreibt J. Ford in seinem Aufsatz »*What is Chaos that we should be mindful for it?*«.

Wo das Chaos beginnt, so die herrschende Meinung, hört die klassische Wissenschaft auf. Solange die Naturwissenschaftler sich nur mit Problemen beschäftigen, die regelmäßige und stetige Lösungen zeigen, *bleiben sie unwissend über die Entwicklung des Wetters, über die Turbulenz von Strömungen, über die Fluktuationen von Tierpopulationen und über die Oszillationen von Herz und Gehirn* (J. Gleick). Besonders die unregelmäßige Seite der Natur wurde als monströse Absurdität abgetan; dies besonders in der Mathematik, wo man die, in keinem Punkt differenzierbaren Kurven von Peano, Hilbert, Koch usw. als Kuriositäten beiseitelegte.

In den siebziger Jahren nahm sich eine kleine Schar von Wissenschaftlern in den USA und Europas der »Unordnung« an. Mathematiker, Physiker, Biologen, Mediziner und Chemiker griffen gemeinsam neue Fragestellungen auf und versuchten, in den verschiedenen Formen von Unregelmäßigkeiten gemeinsame Verbindungen aufzuspüren. Während die Wissenschaft bisher von der immer mehr zunehmenden Spezialisierung profitierte, führte die Chaosforschung dazu, gemeinsame Modelle von dynamischen Systemen zu erarbeiten. In folgenden Gebieten fanden sich u.a. chaotische Modelle:

Wissenschaft	Anwendung
Physik	Erzw.Pendelschwingungen, Turbulenz v.Flüssigkeiten
Optik	Laserphysik
Elektronik	Josephson-Schaltungen

Astronomie	Stabilität der Planetenbahnen
Chemie	Oszillierende chemische Reaktionen
Populationsdynamik	Fluktuationen bei Tierpopulationen

Während etwa ein Drittel aller bei der Zeitschrift *Physical Review Letters* publizierten Arbeiten sich mit Chaos beschäftigen, sind sich die Mathematiker in Europa nicht so einig über den Wert der Chaosforschung und der fraktalen Geometrie. *Das Malen von schönen Apfelmännchen am Bildschirm löst keine ernsthaften Probleme, die Chaos-Mathematik muß ihren Nutzen erst unter Beweis stellen*, ist eine gängige Meinung unter Mathematikern, die der Autor öfters gehört hat.

In Folge sollen einige wichtige Personen und Stationen aufgezeigt werden, die zur Entstehung der Chaostheorie und der Fraktale entscheidend beigetragen haben.

1.1 Henri Poincaré

1887 schrieb der schwedische König Oskar II von Schweden einen wissenschaftlichen Wettbewerb mit dem Preis von 2500 Goldkronen aus. Beantwortet werden sollte die Frage *Ist das Planetensystem stabil*? Den Siegerpreis trug der französische Mathematiker Henri Poincaré davon. Er stellte in seiner Arbeit fest:

Die kanonischen Gleichungen der Himmelsmechanik besitzen kein (außer bei speziellen Anfangsbedingungen) geschlossenes analytisches Lösungsintegral außer dem Energieintegral.

Damit war das Problem zwar nicht gelöst, aber dessen Unlösbarkeit mit analytischen Methoden gezeigt; d.h. mit der Methode von Hamilton-Jacobi. Die Poincaré- Lösung entsprach aber nicht der herkömmlichen Auffassung von Wissenschaft. Poincarés mathematische Untersuchungen, 1899 publiziert als *Les méthodes nouvelles de la mécanique céleste*, bildeten die Grundlage der späteren Theorie der nichtlinearen Systemtheorie. In seinem geistigen Auge sah er den Attraktor eines Drei-Körper-Problems bereits voraus. In Proposition 397 des Werkes schreibt er nämlich:

Wenn man versucht, sich die Figur vorzustellen, die von den beiden Kurven und endlich vielen ihrer Schnittpunkte geformt wird, von denen jeder eine doppelte asymptotische Lösung darstellt, so bilden diese Schnittpunkte eine Art Gitter, Gewebe oder Maschenwerk mit unendlich feinen Maschen. Keine der beiden Kurven darf sich jemals selbst schneiden, sondern muß sich auf sehr komplexe Art so krümmen, daß sie unendlich oft durch alle Maschen des Gitters geht.

Poincarés Theorie wurde später von den Mathematikern Kolmogorow, Arnold und Moser in Form der sog. KAM-Theorie fortgeführt. Von grundlegender Bedeutung

war die Erfindung des Poincaré-Schnitts. Hierbei kann eine Kurve beliebiger Dimension im Phasenraum durch eine Projektion auf eine bestimmte Hyperebene veranschaulicht werden. Poincaré kann – neben David Hilbert – als der letzte Universalist der Mathematik angesehen werden. Er war der Begründer der Topologie und lieferte grundlegende Beiträge u.a. für Differentialgleichungen, Zahlentheorie, komplexe Analysis, Mechanik und Astronomie.

1.2 Balthasar van der Pol

Auch in anderen Zweigen der Wissenschaften kündigten sich die Vorboten des Chaos an. Sie wurden jedoch noch nicht erkannt, da die Zeit noch nicht reif dafür war. Der holländische Elektroingenieur van der Pol hatte 1927 mittels einer Röhrenschaltung einen Oszillator entwickelt (publiziert in *Nature* 120 (1927)). Jeder Physikstudent würde heute das Verhalten eines Schwingkreises am Oszilloskop beobachten. Da es dieses Gerät damals noch nicht gab, verfolgte er die Schwingungen seines Stromkreises mithilfe eines Telefonhörers. Der Ton stabilisierte sich meist kurzzeitig, um dann sprunghaft von einer Frequenz zur nächsten zu wechseln. *Manchmal jedoch*, schreibt er in seinem Bericht an Nature, *verändert sich der Ton in unregelmäßigen Abständen völlig irregulär*. Das Chaos war hier im Telefonhörer hörbar, wurde aber nicht erkannt: *Dies ist jedoch ein untergeordnetes Phänomen,* schreibt er weiter.

Nach dem zweiten Weltkrieg wurde das Verhalten des Van-der-Pol-Oszillators von Cartwright und Littlewood und später von Levinson untersucht. Sie fanden zum erstenmal, daß ein voll determiniertes System irreguläre, damals als zufällig angesehene, Resultate zeigt. Die Ergebnisse des letztgenannten Autors wurden publiziert in: N. Levinson, *A second-order differential equation with singular solutions*, Ann. Math. 50 (1949). Dieser Levinson war es auch, der in einem Brief an S. Smale auf diese Differentialgleichung aufmerksam machte, deren Lösungen zugleich stabil und irregulär waren!

1.3 Edward Lorenz

An einem Wintertag im Jahr 1961 ließ Edward Lorenz am Massachusetts Institute of Technology (MIT) sein zuletzt geschriebenes Konvektions-Programm ein zweites Mal laufen. Jedoch gab er die Anfangswerte, die in FORTRAN standardmäßig sechsstellig ausgegeben werden, verkürzt mit 3 Dezimalen ein, in der Meinung, daß eine Abweichung von einem Promille keine Bedeutung habe.

Zur Berechnung der Luftkonvektion benützte er ein System von 7 Differentialgleichungen, das Barry Saltzmann von der Yale University im gleichen Jahr aus den Navier-Stokes-Gleichungen hergeleitet hatte. Dieses System konnte nicht analytisch

(d.h. formelmäßig) gelöst werden, und man war daher auf eine numerische Computerlösung angewiesen. Lorenz hatte entdeckt, daß bei nichtperiodischen Lösungen der Saltzmann-Gleichungen vier der 7 Variablen gegen Null strebten. Somit konnte Lorenz das System auf drei Gleichungen reduzieren, die heute die Lorenz-Gleichungen genannt werden

Da sein Rechner, ein altes Röhrengerät namens Royal McBee LGP-300, bei einer Rechengeschwindigkeit von 60 Multiplikationen pro Sekunde längere Zeit zur Berechnung benötigte, kehrte er erst nach einiger Zeit zurück, um zu entdecken, daß die zweite Lösung nur noch in den Anfangswerten mit der ersten Lösung übereinstimmte! Alle weitere Werte unterschieden sich beträchtlich voneinander. Lorenz grundlegende Einsicht war es, zu erkennen, daß dies nicht ein Computerfehler, sondern ein prinzipielle Eigenschaft seines Gleichungssystems war.

Dies war die Geburtsstunde eines Phänomens, das Lorenz auf einem wissenschaftlichen Colloqium 1979 in Washington den *Schmetterlingseffekt* nannte: *Der Schlag eines Schmetterlingsflügel über Brasilien kann einen Tornado über Texas hervorrufen.* Schon die geringste Variation in den Anfangsbedingungen eines nichtlinearen Systems ruft eine vollständige Änderung in den Lösungen hervor. Lorenz schreibt:

Ich begriff damals, daß jedes physikalische System, das keine Periodizität aufweist, auch keinerlei Vorhersagen erlaubt.

Diese sensitive Abhängigkeit von den Anfangsbedingungen war bereits von Henri Poincaré vorhergesehen worden. In seinem Werk *La Science et l'Hypothèse* hatte er bereits 1903 geschrieben:

Mitunter kann es geschehen, daß kleine Abweichungen in den Anfangsbedingungen sehr große in den resultierenden Phänomenen hervorrufen. Ein kleiner Fehler in jenem bewirkt einen riesigen Fehler in diesem. Eine Vorhersage wird damit unmöglich.

Lorenz publizierte seine Erkenntnis in dem Artikel *Deterministic Nonperiodic Flow* im Journal of Atmospheric Sciences, 20 (1963), einer meteorologischen Zeitschrift, die von Physikern und Mathematikern kaum gelesen wird. So blieb seine Entdeckung für ein Jahrzehnt weitgehend unbekannt. Die wenigen Naturwissenschaftler, die davon Kenntnis erhielten, wunderten sich nur, wieso ein so einfaches System, wie das der Lorenz-Gleichungen, ein so kompliziertes Lösungsverhalten aufwies. Auch wenn sich die physikalische Bedeutung der Lorenz-Gleichungen nachträglich als gering herausgestellt hat, so sind sie in der Chaostheorie von grundsätzlicher Bedeutung und stellen das in der Literatur am häufigsten untersuchte chaotische System dar.

1.4 Michel Hénon

Der aufschlußreichste, weil einfachste seltsame Attraktor stammt von dem Astronomen Michel Hénon (*A two-dimensional mapping with a strange attractor*, Comm. Math. Phys 50, 1976), der seit 1976 an der Sternwarte Nizza in Südfrankreich tätig war. In seiner Doktorarbeit hatte er 1960, von der willkürlichen Annahme ausgehend, daß ein Sternhaufen bei Maßstabsänderungen selbstähnlich sei, seine Theorie des *gravothermalen Kollaps* geschaffen. Bei einem Studienaufenthalt 1962 in Princeton hatte er zum ersten Mal Zugang zu einem Großrechner.

Damit konnte er ein Problem, wie die Berechnung eines Umlaufs eines galaktischen Systems um ein scheibenförmiges Gravitationszentrum in Angriff nehmen. Dieses nicht analytisch lösbare Problem ist später unter dem Namen *Hénon-Heiles*-Problem bekannt geworden. Wegen der Langsamkeit der damaligen Rechnergeneration mußte Hénon seine Theorie stark vereinfachen. In Ermangelung eines Plotters zeichnete er die Schnittpunkte der galaktischen Bahnen mit einer gedachten Energie-Hyperebene von Hand. Wie alle Astronomen interessierte er sich zunächst nur für periodische Bahnen. Als er jedoch anfing – zusammen mit einem Examensstudenten Carl Heiles – die Schnittpunkte mit höheren Energie-Ebenen zu zeichnen, erkannten beide, daß die Situation sich vollständig wandelte. Sie schrieben im Astronomical Journal 69, (1964) in *The applicability of the third Integral of motion*:

Manche Bahnen wurden so instabil, daß sich die Punkte wie zufällig über das Papier verteilten. Völlige Unordnung trat zusammen mit Resten von Ordnung auf, die an Inseln erinnerten.

Um den zugrundeliegende Mechanismus erkennen zu können, fehlte ihm das mathematische Rüstzeug. Als er 1976 einen Vortrag des Physikers Poumeau über die Attraktoren von Lorenz und Rössler hörte, erhielt er eine Vorstellung, wie es in einem Phasenraum zur Streckung und Faltung kommen kann. Nach dem Vorbild von Smales Hufeisen-Idee fand er durch Kombination von vier einfachen affinen Abbildungen die Gleichungen des nach ihm benannten Attraktors:

$$x_{n+1} = y_n + 1 - 1.4x_n$$
$$y_{n+1} = 0.3x_n$$

1987 konnte Lennard Carleson einen strengen Beweis für die Seltsamkeit des Hénon-Attraktors liefern.

1.5 Robert May

Robert May kam nur durch Zufall zur Biologie. Er hatte sich in Harvard in Angewandter Mathematik habilitiert und wurde nach seinem Weggang nach Princeton 1971 überraschend mit der Abhaltung von Mathematikvorlesungen für Biologen beauftragt.

Er beschäftigte sich unter anderem mit der sog. logistischen Gleichung

$$x_{n+1} = rx_n(1-x_n)$$

deren Namen sich von dem französischen Wort *logis* (Wohnung) ableitet. Diese Gleichung stellt ein Modell für das Wachstum einer Population in Abhängigkeit von dem Parameter r dar. Die logistische Gleichung ist ein Spezialfall der 1845 von Verhulst eingeführten Gleichung

$$N_{t+1} = N_t(a-bN_t)$$

zur Berechnung einer Populationsstärke. Durch graphische Darstellungen erkannte May, daß für kleine Werte von r das Populationswachstum eindeutig ist, für größer werdende Parameter zwischen zwei Populationswerten hin- und herschwankt. Ändert man den Parameter r nur wenig, so oszilliert die Populationszahl nunmehr zwischen 4 Werten, dann zwischen 8, 16, 32 Werten usw., um dann ab einen bestimmten Wert völlig unvorhersagbar zu werden. Dieser Vorgang wird Periodenverdopplung genannt.

Innerhalb dieses chaotischen Bereichs gab es aber wieder Parameterwerte, von May *Fenster* genannt, bei denen sich die Populationszahl auf wenige Werte beschränkte. May erkannte, daß bereits einfache deterministische Modelle ein zufällig erscheinendes, chaotisches Verhalten erklären konnten und daß diese Modelle keineswegs allein auf die Biologie bzw. Populationsdynamik beschränkt waren.

Da schrieb er einen, nach eigenen Worten, *messianischen* Artikel *»Simple Mathematical Models with very Complicated Dynamics«*, der in Nature, 261 (1976) publiziert wurde. Seine Arbeit schließt mit einem Appell, in den Schulen das Rüstzeug zum Verstehen von nichtlinearen Systemen zu lehren:

Nicht allein auf dem Gebiet wissenschaftlicher Forschung, sondern ebenso auf demjenigen der Politik und Wirtschaft stünden wir besser dar, wenn mehr Leute zu der Einsicht kämen, daß einfache nichtlineare Systeme nicht notwendigerweise auch einfache dynamische Eigenschaften zeitigen.

1.6 James Yorke

J. Yorke vom Institute for Physical Science der Universität Maryland gilt als der »Entdecker« von Lorenz Arbeit. Die Geschichte ereignete sich folgendermaßen: Yorke hatte von einem Strömungsspezialisten die Lorenzsche Arbeit erhalten und versandte sie, mit seiner Adresse versehen, an Stephen Smale. Von diesem Exemplar wurde dann eine Vielzahl von Kopien hergestellt, die überall verteilt wurden. So kam es, daß jede Kopie die Anschrift Yorkes trug.

Bei seiner Arbeit an der logistischen Gleichung publizierte er mit seinem Assistenten Li den heute berühmten Artikel mit dem etwas geheimnisvollen Titel *Period Three Implies Chaos* in der mathematischen Zeitschrift mit der größten Auflage, des *American Mathematic Monthly*. Dieser Aufsatz gab der Chaostheorie ihren Namen.

Anläßlich eines mathematischen Kongresses im (damals noch zweigeteilten) Berlin machte Yorke eine Wannsee-Dampferfahrt. Dabei wurde er von einem, ihm unbekannten, russischen Mathematiker bedrängt, der ihm mit Hilfe eines englisch sprechenden, polnischen Mathematikers zu erklären versuchte, daß seine Entdeckung keineswegs neu sei, genauere Details würde er ihm noch mitteilen.

Vier Monate später erhielt Yorke von A. N. Sarkowskii – dies war der Name des russischen Mathematikers – einen Artikel »*Coexistence of cycles of a continous map of a line itself*«, in dem Yorkes Ergebnis in verallgemeinerter Form vorweggenommen war. Sarkowsky hatte nämlich eine spezielle Anordnung der natürlichen Zahlen gefunden, die es ermöglichte, aufgrund einer bekannten Periode auch alle weiteren Periodenlängen anzugeben. Interessant ist, daß dabei nur die Stetigkeit der verwendeten Funktion eingeht. Allerdings gilt das Theorem nur für eindimensionale Abbildungen.

1.7 Stephen Smale

S. Smale von der Berkley-Universität (California) hatte sich 1966 bereits internationalen Ruhm auf Grund seiner Arbeiten über mehrdimensionale Topologie erworben. Er hatte ein Problem über Mannigfaltigen in mindestens 5 Dimensionen gelöst, das 1906 von Poincaré aufgeworfen worden war und als unangreifbar galt. Smale betrachtete aber die Wissenschaft nicht als Elfenbeinturm und engagierte sich u.a. stark beim Protest gegen Vietnamkrieg.

So erhielt er im Sommer desselben Jahres eine Vorladung vor das Kommitee für antiamerikanische Umtriebe, was ihn aber nicht hinderte, nach Moskau zum Internationalen Mathematikerkongreß zu fahren. Dort konnte er die Fields-Medaille, die höchste mathematische Auszeichnung in Empfang nehmen. Als er auf der Haupttreppe der Moskauer Universität stehend, eine internationale Presseerklärung gegen

den Vietnamkrieg – aber auch gegen jede sowjetische Aggression – abgab, wurde ihm daraufhin vom Nationalen Wissenschaftsrat sein Forschungsauftrag entzogen.

Seit langem hatte sich Smale intensiv mit der Lösbarkeit von Differentialgleichungen auseinandergesetzt. 1959 erreichte ihn dann der Brief von Levinson, der ihm zeigte, daß Systeme – wie der Van-der-Pol-Oszillator – durchaus stabile und chaotische Lösungen haben können. Um das Verhalten des Oszillators verstehen zu können, untersuchte er die Transformationen, die der Phasenraum des Oszillators erfährt. Dabei erkannte er, daß es zur Erklärung des chaotischen Verhaltens nicht ausreicht, nur die Dehnung und Stauchung des Phasenraum zuzulassen. Vielmehr werden auch Faltungen benötigt, um den Weg ins Chaos zu beschreiben. Dieser Mechanismus wurde als Hufeisen-Modell (englisch *horseshoe*) bekannt.

Mit Hilfe der von Smale entwickelten symbolischen Dynamik können dynamische Systeme als äquivalent klassifiziert werden.

1.8 Mitchell Feigenbaum

Außer May beschäftigte sich eine ganze Reihe weiterer Forscher mit der logistischen Gleichung. Darunter befanden sich S. Großmann und S. Thomae von der Universität Marburg und später P. Collet und J. Eckmann. Das erstgenannte deutsch-französische Autorenpaar entdeckte das geometrische Gesetz der Periodenverdopplung der logistischen Gleichung und berechnete bereits die Konstante, die nun allgemein die Feigenbaum-Konstante genannt wird. S. Thomae ist übrigens der Begründer der Katastrophentheorie. Mit ihrer Publikation in der Zeitschrift für Naturforschung A32a (1977) kamen Großmann und Thomae der Publikation von M. Feigenbaum in J. Stat. Phys. 19 (1978) zuvor.

M. Feigenbaum, der durch keine spezielle Publikation bisher hervorgetreten war, beschäftigte sich aufgrund eines Referats von S. Smale im Sommer 1975 ausschließlich mit chaotischen Problemen, speziell aber mit der Periodenverdopplung der logistischen Gleichung.

Von den Kollegen N. Metropolis und P. und M. Stein erfuhr er, daß auch andere Funktionen wie $x_{n+1} = r\sin(\pi x_n)$ das Phänomen der Periodenverdopplungen zeigten. Als er für diese Gleichung dieselbe Konstante $F = 4.6692016090..$ wie für die logistische Gleichung ermittelte, traf ihn wie ein Blitz die Erkenntnis, daß es sich hier um eine universelle Beziehung handeln müsse.

Dieses universelle Gesetz konnte er nicht beweisen, sondern nur durch zahlreiche numerische Berechnungen motivieren. Die meisten Mathematiker blieben wegen des fehlenden Beweises skeptisch, unter anderen insbesondere auch Mandelbrot; die Physiker waren begeistert. R. Gilmore schreibt

Es war eine ebenso glückliche wie schockierende Entdeckung, daß es in nichtlinearen Systemen Strukturen gibt, die immer gleich bleiben, wenn man sie nur in der richtigen Weise ansieht.

Einen ersten Beweis lieferte dann 1979 O. Lanford, der später in der Arbeit »*A computer-assisted proof of the Feigenbaum conjectures*« publiziert wurde. Erst später gelang es Feigenbaum, eine Funktionalgleichung für die universelle Funktion mit Periodenverdopplungen aufzustellen, aus der dann die Feigenbaum-Konstante als Eigenwert erschien. Diese Funktionalgleichung war:

$$g(x) = -\alpha g(g(\tfrac{x}{\alpha}))$$

Auch wenn Feigenbaum von den Mathematikern nicht die gewünschte Anerkennung erhielt, so wird doch sein Namen mit den Gabel-Bifurkationen und Bifurkations-Diagrammen verbunden sein.

1.9 David Ruelle

D. Ruelle, von Geburt Belgier, war 1970 zum Institut des Hautes Études Scientifique gekommen. Er hatte in Vorlesungen von Stephen Smale und dessen Hufeisenabbildung gehört. Seitdem trug er sich mit dem Gedanken, daß das Auftreten von Turbulenzen in Flüssigkeiten eine Auswirkung des Chaos sein müsse. Zusammen mit dem damaligen Gast am Institut, F. Takens, schrieb er einen Aufsatz *On the Nature of Turbulence*, in dem Turbulenzen im Sinne von Smale als Dehnungen und Faltungen des Phasenraums, erklärt wurden. Die Geometrie des Vorgangs nannte er einen seltsamen Attraktor (englisch *strange attractor*) und prägte damit einen grundlegenden Begriff der entstehenden Chaosforschung.

Die meisten Physiker lehnten damals die neue Theorie ab. Als Beispiel sei der berühmte Theoretiker R. Feymann zitiert:

Es hat mich stets irritiert, daß den Gesetzen zufolge, soweit wir sie heute verstehen, ein Computer eine unendlich große Zahl lokaler Operationen durchführen muß, um auszurechnen, was in einer winzigen Raumzone vor sich geht in einem beliebig kurzen Zeitabschnitt.

Auch heute haben viele Physiker Probleme, sich einen solchen Attraktor vorzustellen. Er soll stabil sein – um einen Endzustand eines dynamischen Systems darzustellen. Er soll niedrigdimensional sein – damit er eine Bahn in einem Phasenraum von wenigen Freiheitsgraden bestimmt. Und er soll nichtperiodisch sein und dennoch das ganze Spektrum der bei Turbulenzen auftretenden Frequenzen erklären. Mit einem Wort, der Attraktor mußte fraktal sein. Der von Ruelle und Takens vorgeschlagene Weg ins Chaos ist als *Ruelle-Takens-Szenario* in die Literatur eingegangen.

Einen dritten Weg ins Chaos fanden Manneville und Poumeau 1979 in Form der sog. Intermittenz-Route. Intermittenz (englisch *intermittence*) bedeutet dasjenige Verhalten eines Systems, das sich weitgehend regulär verhält (entsprechend der laminaren Phase einer Strömung), aber durch statistisch verteilte irreguläre Perioden unterbrochen wird. Diese irreguläre Störungen nehmen zu bis zu einem Punkt, an dem das Verhalten des Systems ins Chaos übergeht; analog dem Übergang einer laminaren Strömung in eine Turbulenz. Bei diesem Verhalten spricht man nun vom *Manneville-Poumeau-Szenario*.

1.10 Benoît Mandelbrot

»*Hüte Dich vor der Geometrie*« sagte der Mathematiker Scholem Mandelbrojt seinem Neffen Benoît, als dieser in Paris Mathematik studieren wollte. Er empfahl ihm das Studium einer 300-seitigen Arbeit von Gaston Julia aus dem Bereich der komplexen Analysis. Aber diese Arbeit interessierte den damals noch jungen Benoît Mandelbrot nicht, obwohl er bei G. Julia Vorlesungen über Differentialgeometrie gehört hatte. *Noch* nicht, muß man sagen, denn das Werk von G. Julia und P. Fatou sollte später bestimmend für seine Arbeit sein.

Als Mandelbrot 1958 zum IBM-Forschungsinstitut Thomas J. Watson in Yorktown Heights (New York) kam, war er ein Mann für alle Aufgaben. Er beschäftigte sich mit volkswirtschaftlichen Problemen wie die Entwicklung der Baumwollpreise, tätigte Untersuchungen über Störgeräusche in Telefonnetzen und stellte ein Gesetz über die Worthäufigkeiten im Englischen auf. Nebenbei studierte er eine Vielzahl von unkonventionellen Arbeiten aus den verschiedensten mathematischen Themenbereichen. Durch Zufall (oder *Schicksal* wie Mandelbrot schrieb) fiel ihm ein völlig unbekanntes statistisches Jahrbuch von 1961 in die Hand. Darin fand er einen Aufsatz von einem L. F. Richardson, der sich mit der Problematik der Längenmessung von Küstenlinien befaßte. Die Beschäftigung mit diesem Aufsatz wurde zum entscheidenden Wendepunkt in seinem Denken.

Er erkannte schnell, daß die widersprüchlichen Werte für Küstenlängen, die in Enzyklopädien und Lexika mit Abweichungen bis zu 20% angegeben werden, auf die Verwendung verschiedener Maßstäbe zurückzuführen sind. Je feiner der angewandte Maßstab ist, umso mehr Details mißt er. Denkt man sich die Küstenlinie beliebig vergrößert, so könnte bei einem sehr kleinen Maßstab die Küstenlänge beliebig lang werden, schloß Mandelbrot.

Bei der Untersuchung der Cantor-Menge, den Kurven von Hilbert, Peano und Sierpinski erkannte er, daß diese Kurven nicht die Dimension einer Geraden haben konnten. Für diese Art von Objekten bildete er im Winter 1975 bei den Vorarbeiten für sein Buch *Les objects fractal: forme, hasard et dimension* den Namen Fraktale. Dies

war eine Ableitung vom lateinischen Wort *frangere* für brechen mit dem Partizip *fractus*. Mandelbrot, der zum Ärger der anderen Mathematiker stets neue Begriffe erfand, nannte den Dimensionsbegriff von F. Hausdorff und A. Besikowitsch fraktal und vereinnahmte damit den Begriff für sich.

Die nicht von ihm allein entwickelte Theorie der Selbstähnlichkeit und Skaleninvarianz konnte auf eine Vielzahl von Objekten angewendet werden – von der Geometrie der Mondkrater und Verteilung der Galaxienhaufen bis hin zum Verlauf von Küstenlinien. Auf der Grundlage des Skalenverhaltens der Brownschen Bewegung konnte Mandelbrots Mitarbeiter R. Voss atemberaubende Graphiken von Landschaften mit Bergketten, Tälern und Wolken produzieren. Dies rief auch bald die Filmindustrie auf den Plan. L. Carpenter von Boeing Aircraft, später bei Lucasfilm, entwickelte die Graphik-Techniken von Voss weiter und konnte so alle Hintergrundbilder des Films *Startrek* II (Wrath of Khan) am Bildschirm erzeugen.

In den Jahre 1979-80 war Mandelbrot Visiting-Professor für Mathematik an der Harvard Univerity und nahm seine Arbeit am Werk von Julia und Fatou wieder auf. Da plötzlich, genau am 1. April 1980, erschien eine apfelähnliche Figur am Bildschirm. Die Iteration der einfachen komplexen Gleichung $z_{n+1} = z_n^2 + c$ war der Pfad zur Entdeckung des Apfelmännchens, das heute zum Symbol der fraktalen Geometrie geworden ist. Alle seine Begriffe und Definitionen schrieb Mandelbrot 1982 in seinem berühmten Buch *The fractal Geometry of Nature* nieder, das als die Bibel der fraktalen Geometer angesehen wird. Es wurde angeblich zum meist verkauften Mathematikbuch aller Zeiten.

1.11 Michael Barnsley

Barnsley, der in Oxford studiert hatte, kam 1979 durch ein Treffen mit M. Feigenbaum zur Kenntnis der Bifurkationen und Periodenverdopplungen. Voller Eifer machte er sich ans Werk, das Phänomen der Periodenverdopplung mit Hilfe komplexer Methoden zu erklären. Sein Aufsatz, den er bei der Zeitschrift *Communications in Mathematical Physics* eingereicht hatte, erhielt er vom Herausgeber David Ruelle zurück mit der Bemerkung, Julia-Mengen seien wohlbekannt, er solle sich mit Mandelbrot in Verbindung setzen.

Bei der SIGGRAPH-Tagung 1985 in Kalifornien erregte Barnsley großes Aufsehen. Es war ihm und seinem Team mit Hilfe von vier einfachen affinen Abbildungen gelungen, ein realistisches Bild von einem Farnkraut zu erzeugen. Barnsley sagte bei der Entdeckung:

Das Bild war verblüffend, es stimmte in allen Einzelheiten mit dem Original überein. Kein Biologe hätte Schwierigkeiten, den Farn zu bestimmen.

Das Verfahren hieß *Iteriertes Funktionssystem* (IFS) und wurde in den *Proceedings of the Royal Society of London*, A 399 (1975) von den Autoren M. Barnsley und S. Demko publiziert.

Barnsley nennt die Methode das *Chaosspiel*. Gleick erklärt es durch folgenden Vergleich. Angenommen, man will eine fraktale Küstenlinie nachbilden. Die Küstenlinie wird mit Kreide auf den Boden gezeichnet und darauf eine Vielzahl von Reiskörner zufällig gestreut. Zählt man nur die Reiskörner, die auf die gezeichneten Linien fallen, so bilden diese bei Fortsetzung des Verfahrens genau den Küstenverlauf ab. Genaugesehen sind Barnsleys Formen, wie der erwähnte Farn, Attraktoren.

Das Chaosspiel macht sich die fraktalen Eigenschaften bestimmter Bilder zunutze. Diese selbstähnlichen Bilder bestehen wiederum aus verkleinerten Kopien des ganzen Bildes. Bestimmt man nun die mathematischen Transformationen, durch die die verkleinerten Kopien aus dem Ganzen hervorgehen, so ist das Iterierte Funktionssystem (IFS) damit eindeutig bestimmt. Mit Hilfe des von Barnsley gefundenen Collage-Theorems kann man das IFS auch von komplizierten Abbildungen bestimmen.

Auch das umgekehrte Problem ist interessant: Wie findet man zu einem fertigen Bild die Gleichungen eines zugehörigen IFS? Barnsley gelang es, dafür einen maschinengeeigneten Algorithmus zu finden und ihn patentieren zu lassen. Mathematische Algorithmen sind in den USA, im Gegensatz zur Bundesrepublik, patentfähig. Ebenso wie das komplizierte Farnblatt auf 4 Abbildungen mit je 6 Parametern zurückgeführt werden kann, so kann der – inzwischen patentierte – Algorithmus von Barnsley ein beliebiges Bild auf seinen IFS-Code reduzieren. Diese Methode scheint eine der vielversprechendsten Methoden der Datenkompression für Bilder zu sein.

1.12 Aristid Lindenmayer

Aristid Lindenmayer war theoretischer Biologe an der Universität von Utrecht. Sein Betreben war es, die Entwicklung von Pflanzen mit Hilfe mathematischer Algorithmen kennzeichnen zu können. Lindenmayer schreibt in einem 1975 erschienen Aufsatz

Die Entwicklung eines Organismus kann angesehen werden als Ausführung eines »Entwicklungsprogramms«, das sich in der befruchteten Eizelle befindet. ... Eine zentrale Aufgabe der Entwicklungsbiologie ist es, die zugrundeliegenden Algorithmen aus dem Lauf der Entwicklung herzuleiten.

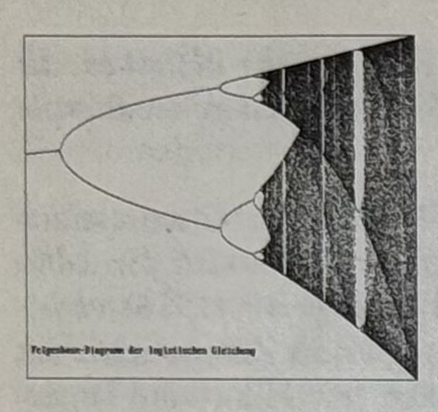
Besonders beeindruckte ihn das Prinzip der Selbstähnlichkeit, das er bei Mandelbrot gefunden hatte. In einem anderen Artikel desselben Jahres schreibt er:

Bei vielen Wachstumsprozessen von lebenden Organismen, besonders bei Pflanzen, sind wiederholte regelmäßige Erscheinungen von gewissen mehrzelligen Strukturen

erkennbar. ... Im Fall eines zusammengesetzten Blattes z.B. haben die Blättchen, die Teil eines Blatts in einem späteren Zustand sind, dieselbe Form wie das ganze Blatt in einem früheren Zustand.

Auf der Grundlage der von N. Chomsky geschaffenen Theorie der Grammatiken konnte Lindenmayer 1986 sein grundlegendes Werk *Mathematical models for cellular interaction in development* im Journal of Theoretical Biology 18 (1986) veröffentlichen. P. Prusinkiewz von der Universität Regina gelang es, in Zusammenarbeit mit Lindenmayer, die mathematischen Graph-Grammatiken mit Hilfe von Turtle-Graphik in Computeralgorithmen umzusetzen. So entstand der bekannte Formalismus, der nun Lindenmayer- oder kurz *L-Systeme* genannt wird.

Die auf der SIGGRAPH 1988 und 1989 gehaltenen Vorträge wurden in den Lecture Notes in Biomathematics publiziert. Daraus entstand 1990 der berühmte Band »*The Algorithmic Beauty of Plants*«, der von vielen Leuten als das schönste Buch mit mathematischen Formeln angesehen wird. Schade, daß Lindenmayer das Erscheinen seines Buches nicht mehr erlebt hat.



2 Die logistische Gleichung

Die logistische Gleichung (*logistic map*)

$$x_{n+1} = rx_n(1 - x_n)$$

ist ein Spezialfall der 1845 von P. F. Verhulst zur Beschreibung eines Populationswachstums eingeführten Gleichung

$$N_{t+1} = N_t(a - bN_t)$$

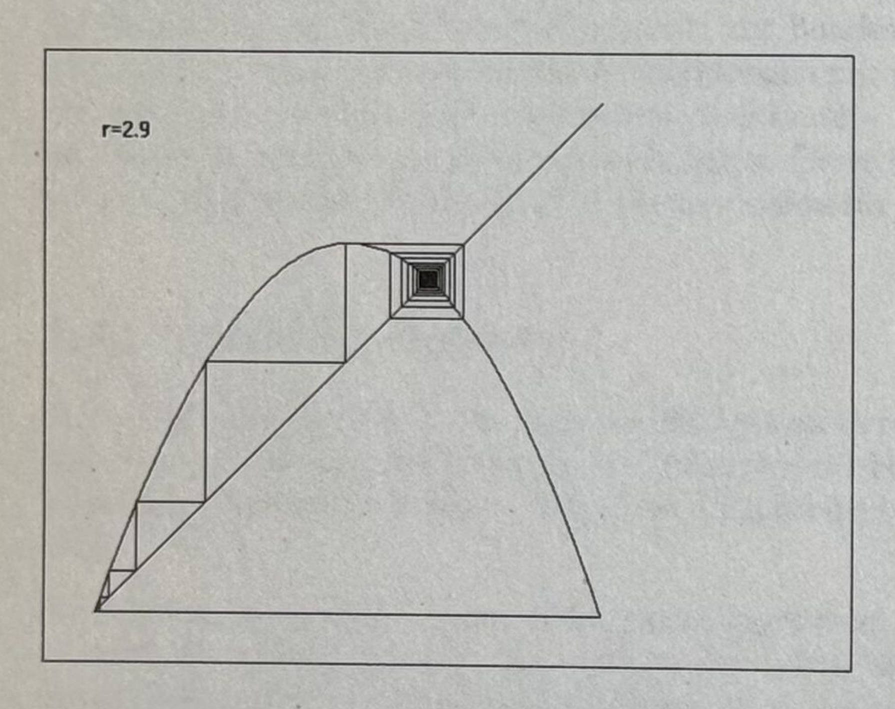


Bild 2.1 Iteration der logistischen Gleichung für r=2.9

Auf der Suche nach einem deterministischen Zufallszahlen-Generator, wurde die Gleichung 1947 von S. M. Ulam und J. von Neumann intensiv studiert. Jedoch zeigte sich, daß die mittels der logistischen Gleichung erzeugten Zahlen im Intervall [0;1] nicht genügend gleichverteilt waren (vgl. Bild 2.6).

Mit dem Artikel »*Simple mathematical models with very complicated dynamics*« in Nature Volume 261, 1976 entdeckte Robert M. May die logistische Gleichung für die deterministische Chaostheorie. Er untersuchte die Dynamik der Gleichung für die verschiedenen Werte des Parameters $r \in [0;4]$. Das Phänomen der Periodenverdopplung wurde 1977 von S. Großmann und S. Thomae und 1978 von Mitchell Feigenbaum gefunden.

Die logistische Gleichung stellt keine spezielle quadratische Gleichung dar. Die allgemeine quadratische Gleichung $x_{n+1}^2 = A + Bx_n + Cx_n^2$ geht durch die Transformation $x_n \to kx_n + d$ über in die Form

$$x_{n+1} = (A - d + Bd + Cd^2)/k + (B + 2Cd)x_n + Ckx_n^2$$

Dies ist äquivalent zu $x_{n+1} = rx_n(1 - x_n)$ mit

$$A = \frac{r}{4} + \frac{1}{2};\ B = 0;\ C = -r;\ k = 1;\ d = \frac{1}{2}$$

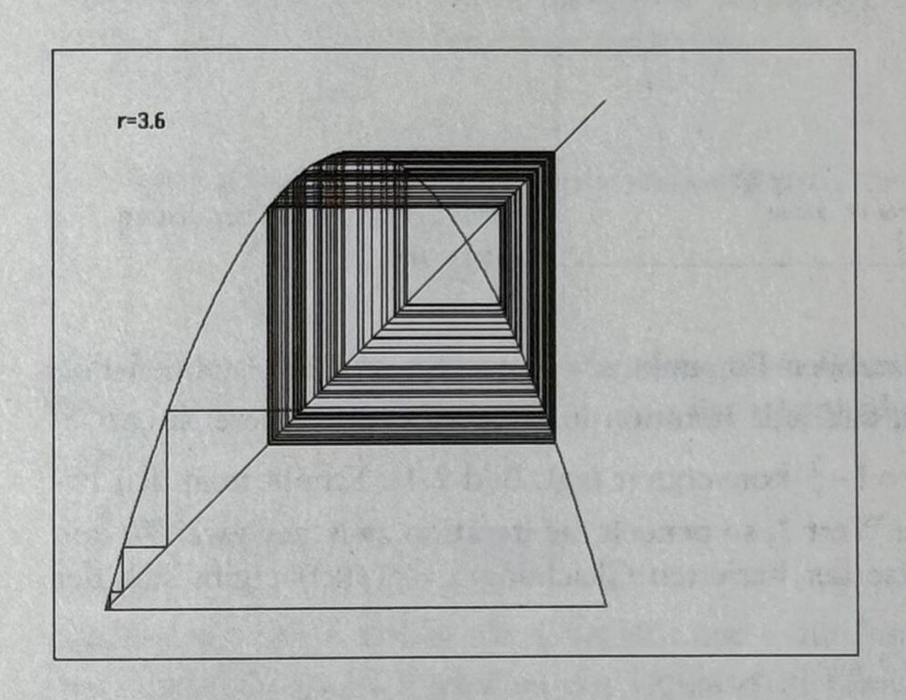


Bild 2.2 Iteration der logistischen Gleichung für $r = 3.6$

2.1 Dynamik der logistischen Gleichung

Fixpunkte einer Gleichung sind definiert durch die Gleichung

$$f(x) = x$$

Ein Fixpunkt heißt stabil oder anziehend, wenn gilt

$$|f'(x)| < 1$$

superstabil für

$$|f'(x)| = 0$$

und instabil oder abstoßend für

$$|f'(x)| > 1$$

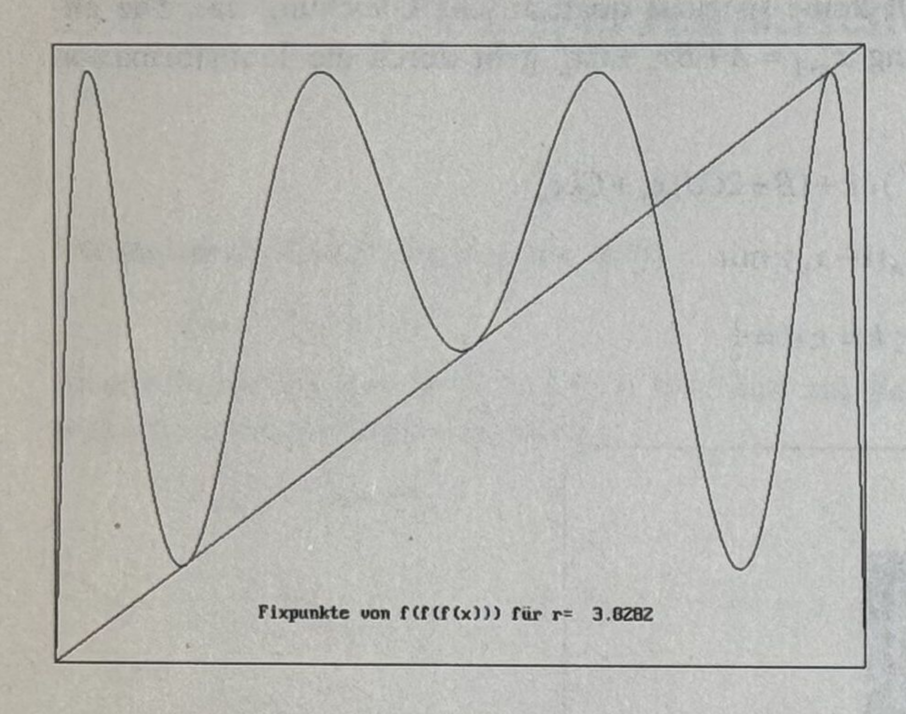


Bild 2.3 Fixpunkte der dritten Iterierten

Für $0 < r \leq 1$ erhält man den stabilen Fixpunkt $x = 0$, für $1 < r \leq 3$ den stabilen Fixpunkt $x = 1 - \frac{1}{r}$. Dies bedeutet, daß jede Iteration in diesem Parameterbereich entweder gegen $x = 0$ oder gegen $x = 1 - \frac{1}{r}$ konvergiert (vgl. Bild 2.1). Erhöht man den Parameter r geringfügig über den Wert 3, so pendelt die Iteration zwischen zwei Werten hin und her. Für die Fixpunkte der iterierten Gleichung $x = f(f(x))$ ergibt sich der Ansatz

$$x_2 = rx_1(1 - x_1)$$
$$x_1 = rx_2(1 - x_2)$$

Neben der trivialen Lösung $x_1 = x_2$ erhält man den Zweierzyklus

$$x = \tfrac{1}{2}(1 + \tfrac{1}{r}) \pm \tfrac{1}{2r}\sqrt{r^2 - 2r - 3}$$

für $3 < r \leq 1 + \sqrt{6}$. Gegenüber der Konvergenz gegen einen Fixpunkt hat sich hier die Anzahl der Fixpunkte verdoppelt; man spricht daher von *Periodenverdopplung*.

In ähnlicher Weise kann man die Fixpunkte der dritten und vierten Iterierten bestimmen (in Bild 2.3 als Schnittpunkt mit der Winkelhalbierenden des 1. Quadranten zu erkennen). Schreibt man die Parameterwerte, bei denen je eine Periodenverdopplung auftritt, in Form einer Tabelle an, so erhält man

Parameterwerte mit Bifurkationen
$r_1 = 3$
$r2 = 3.449499$
$r3 = 3.544090$
$r4 = 3.564407$
$r5 = 3.568759$
$r6 = 3.569692$
$r7 = 3.569891$
$r_8 = 3.569934$

Diese Folge scheint sich einem Grenzwert zu nähern. Macht man den Ansatz einer geometrischen Folge mit dem Skalenverhalten

$$r_k = r_\infty - cF^{-k}$$

so erhält man mittels eines Extrapolationsverfahrens die Werte $c = 2.6327$ und

$$F = \frac{r_{k+1} - r_k}{r_{k+2} - r_{k+1}} = 4.669202$$

Die letztere Konstante wird nun *Feigenbaum-Konstante* genannt. Mit Hilfe dieses Wertes läßt sich der Grenzwert r_∞ der Periodenverdoppelungen extrapolieren. Es ergibt sich

$$r_\infty = \frac{Fr_{k+1} - r_k}{F - 1} = 3.5699456$$

Für Parameterwerte größer als r_∞ beginnt nun – mit Ausnahme der periodischen Fenster – das chaotische Verhalten der logistischen Gleichung; d.h. der Wertebereich wächst stark an und füllt für $r = 4$ alle Werte des Intervalls [0;1]. In den periodischen Fenstern treten auch Zyklen der Periodenlängen 3,5,6 usw. mit den zugehörigen Periodenverdopplungen auf (vgl. Abbildung 2.4).

Von besonderer Bedeutung ist ein auftretender Dreierzyklus, da nach dem berühmten Satz von Yorke und Li (1975) die Periode drei chaotisches Verhalten nach sich zieht. Der Satz ist ein Spezialfall des Satzes von Sarkowskii (Siehe Kapitel 19).

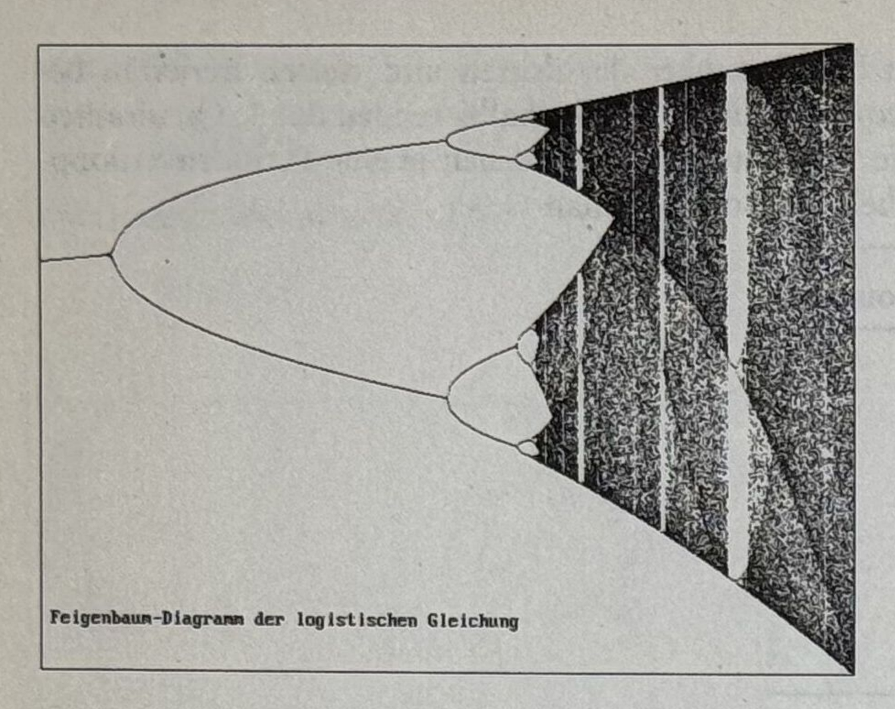


Bild 2.4 Feigenbaum-Diagramm der logistischen Gleichung

Die Feigenbaum-Konstante F ist universell in dem Sinn, daß sie für alle unimodalen Funktionen mit einem quadratischen Maximum (d.h. $f''(x_m) \neq 0$) gilt. Beispiele für solche Funktionen sind

$$f(x) = r\sin(\pi x) \qquad f(x) = r\sqrt{x(1-x)}$$
$$f(x) = rx(1-x)^2 \qquad f(x) = rx^2\sin(\pi x)$$

Auch bei vielen physikalischen und chemischen Versuchen wie bei der Josephson-Schaltung und beim Brüsselator, tritt die Feigenbaum-Konstante auf. Einen guten Überblick über das Auftreten der Feigenbaum-Konstanten in den Naturwissenschaften liefert das Vorwort von P. Cvitanovic im Sammelband [05].

2.3 Das Feigenbaum-Diagramm

Das dynamische Verhalten der Funktion kann mittels des Feigenbaum-Diagramms veranschaulicht werden. Dabei werden die Werte des Parameters r auf der waagerechten Achse, die Funktionswerte auf der senkrechten Achse angetragen. Die Periodenverdopplung zeigt sich an der »Gabelung« der Funktionswerte, auch Feigenbaum- oder Gabel-Bifurkation genannt.

Mit dem Quick Basic-Programm `feigenb.bas` kann das Feigenbaum-Diagramm der logistischen Gleichung erstellt werden.

```
'feigenb.bas

SCREEN 12: CLS
WINDOW (2.9, 0)-(4, 1)
```

```
LINE (2.9, 0)-(4, 1), , B

FOR r = 2.9 TO 4 STEP 1.1 / 640
   x = RND
   FOR i = 1 TO 700
      x = r * x * (1 - x)
      IF i > 400 THEN PSET (r, x), 9
   NEXT i
NEXT r
LOCATE 28, 2: PRINT "Feigenbaum-Diagramm der logistischen Gleichung"
e$ = INPUT$(1): SCREEN 0
END
```

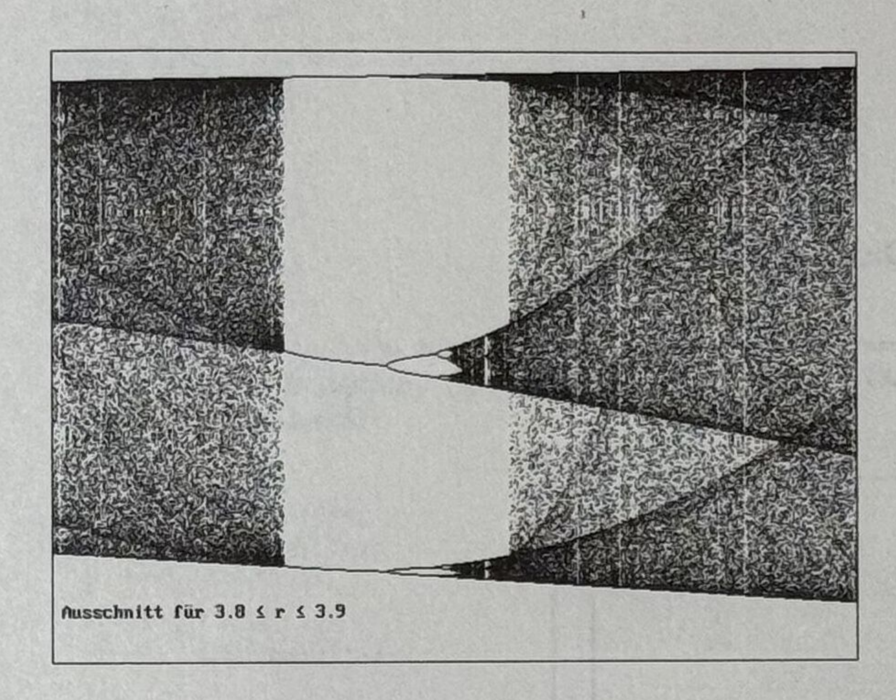


Bild 2.5 Ausschnitt aus dem Bifurkations-Diagramm mit Dreierzyklus-Fenster

2.4 Exakte Lösung im Chaos-Fall

Interessanterweise läßt sich im Fall $r = 4$ eine exakte Lösung der logistischen Gleichung angeben. Wie man durch Einsetzen verifizieren kann, gilt diese Lösung

$$x_n = \frac{1}{2} - \frac{1}{2}\cos(2^n \arccos(1 - 2x_0))$$

für alle Anfangswerte

$$x_0 \neq \frac{1}{2} - \frac{1}{2}\cos\frac{r\pi}{2^s};\, r, s \in Z.$$

2.5 Die invariante Dichte

Wie in der Statistik kann für die viele chaotische Funktionen eine Dichtefunktion berechnet werden. Das Differential

$$\rho(x)dx = d\xi$$

stellt die Wahrscheinlichkeit dar, Funktionswerte im Intervall $[x; x+dx]$anzutreffen. Für die Substitution

$$x = \sin^2 \frac{\pi\xi}{2}$$

folgt durch Umkehrung

$$\xi = \frac{2}{\pi} \arcsin \sqrt{x}$$

Einsetzen ergibt die gesuchte Dichtefunktion

$$\rho(x) = \frac{d\xi}{dx} = \frac{1}{\pi} \frac{1}{\sqrt{x(1-x)}}$$

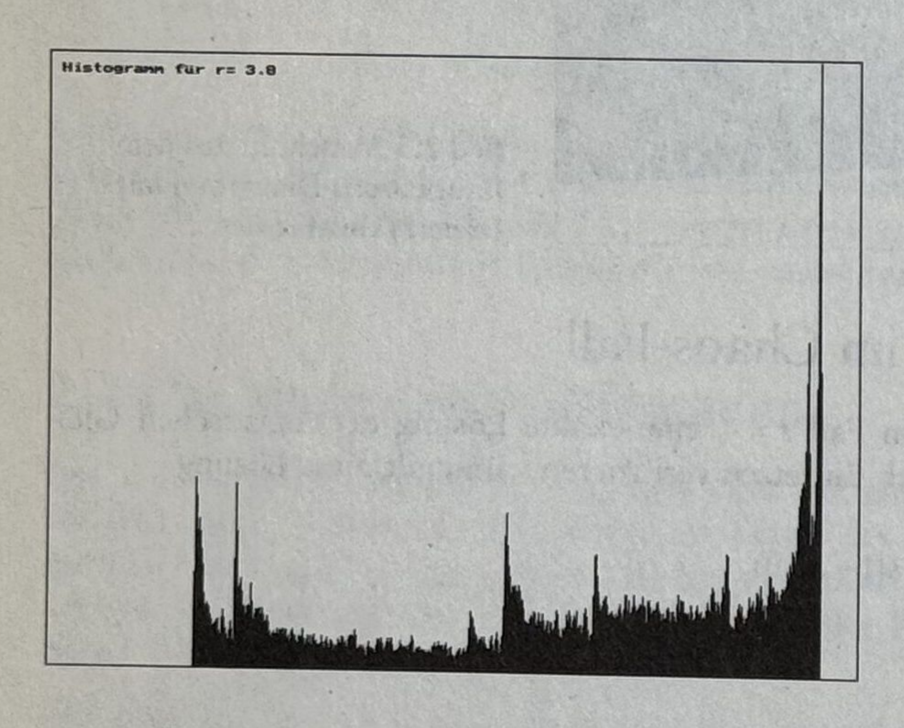


Bild 2.6 Histogramm der logistischen Funktion für r=3.8

Es läßt sich auch eine geschlossene Formel für die Dichtefunktion angegeben

$$\rho(x) = \lim_{n\to\infty} \frac{1}{n} \sum_{i=0}^{n} \delta\,(x - x_i)$$

wobei δ die berühmte Diracsche Deltafunktion ist. Diese Definition ist aber für praktische Zwecke weniger geeignet. Mit dem Turbo C-Programm `histogr.c` erhält man das Histogramm der logistischen Kurve.

```
/* histogr.c */

#include <stdio.h>
#include <graphics.h>
#include <stdlib.h>
#include <math.h>
#include <conio.h>

const double pi = 3.141529653;
const int n = 640;
const double r = 3.8;
static int h[640];

void main(void)
{
int gdriver,gmode;
int max,i;
double x=pi/4.;

detectgraph(&gdriver,&gmode);
initgraph(&gdriver,&gmode,"\\bc\\bgi");
rectangle(0,0,639,479);
setcolor(12);

h[(int) floor(n*x)]++;
for (i=0; i<20000; i++)
  {
  x = r*x*(1.-x);
  h[(int) floor(n*x)]++;
  }
for (i=0,max=0; i<n; i++)
     if (h[i]>max) max= h[i];
moveto(1,480-(int)(478./max*h[0]));
for (i=1; i<n; i++)
   lineto(i+1,480-(int)(478./max*h[i]));
moveto(10,10);outtext("Histogramm für r= 3.8");
getch();
closegraph();
}
```

Wie man leicht nachprüft, erfüllt diese Funktion die in der Statistik geforderte Normierungsbedingung

$$\int_0^1 \rho(x)dx = 1$$

vermöge des Integrals

$$\frac{1}{\pi}\int_0^1 \frac{1}{\sqrt{x(1-x)}}dx = \frac{1}{\pi}\Big[\arcsin(2x-1)\Big]_0^1 = 1$$

Nach Lauwerier (*One dimensional iterative maps* im Sammelband [13]) läßt sich damit der (maximale) Ljapunow-Exponent λ berechnen

$$\lambda = \int \ln\left|\frac{df}{dx}\right| \rho(x)dx$$

Für die logistische Funktion folgt mit $f'(x) = 4(1-2x)$

$$l = \frac{1}{p}\int_0^1 \frac{\ln|4(1-2x)|}{\sqrt{x(1-x)}}dx = \ln 2$$

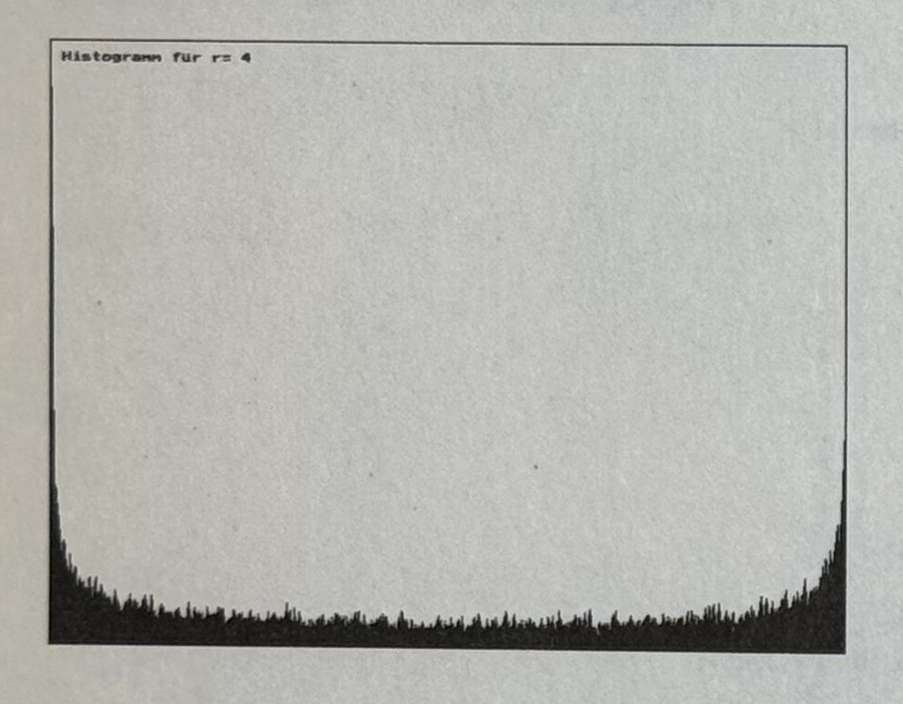


Bild 2.7 Histogramm der logistischen Funktion für r=4

2.6 Berechnung des Ljapunow-Exponenten

Der Ljapunow-Exponent ist ein Maß für den Abstand zweier benachbarter Bahnen $\left(x_n \middle| y_n\right)$ mit $y_n = x_n + \delta x_n$

Für y_n folgt nach dem Mittelwertsatz der Differentialrechnung

$$y_{n+1} = f(y_n) = f(x_n + \delta x_n) \approx f(x_n) + f'(x_n)\delta x_n$$

Der Vergleich mit

$$y_{n+1} = x_{n+1} + \delta x_{n+1} = f(x_n) + \delta x_{n+1}$$

liefert die Gleichung

$$\delta x_{n+1} = \mu \delta x_n$$

mit $\mu = |f'(x_n)|$.

Durch Iteration erhält man

$$\delta x_n = \mu^n \delta x_0$$

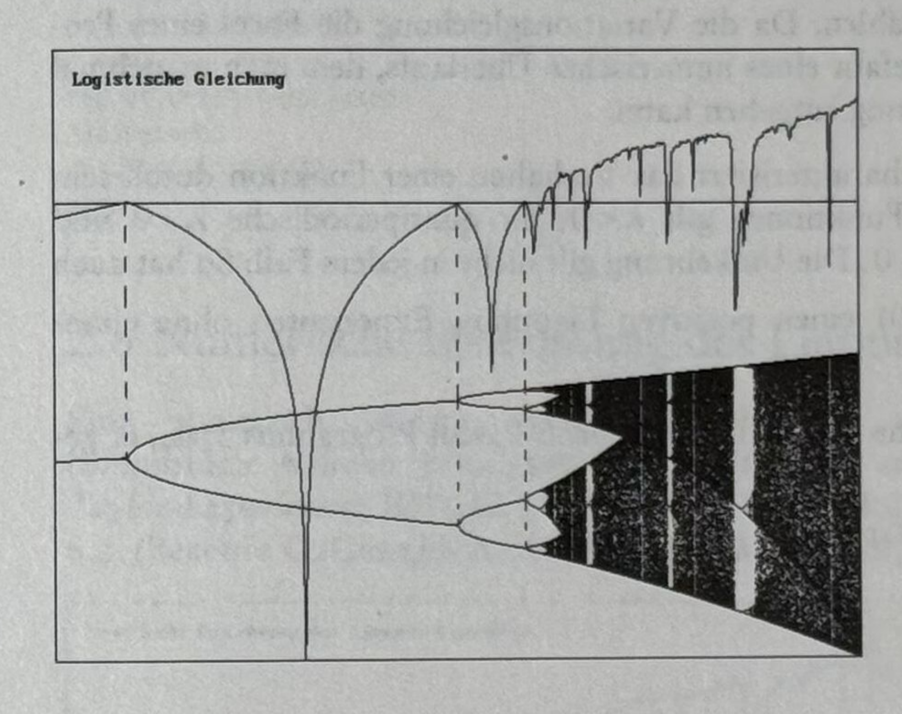


Bild 2.8 Ljapunow-Exponent im Intervall [2.8; 4]

Der Ansatz eines exponentiellen Wachstums

$$\delta x_n = e^{\lambda n} \delta x_0 = \prod_{i=0}^{n-1} \mu_i \delta x_0 \quad \text{liefert}$$

$$\ln(e^{\lambda n}) = \lambda n = \ln\left(\prod_{i=0}^{n-1} \mu_i\right) = \sum_{i=0}^{n-1} \ln(\mu_i) = \sum_{i=0}^{n-1} \ln|f'(x_i)|$$

Der (maximale) Ljapunow-Exponent ergibt sich damit aus dem Grenzwert

$$\lambda = \lim_{n\to\infty} \frac{1}{n} \sum_{i=0}^{n-1} \ln|f'(x_i)|$$

Diese Form ist jedoch für numerische Zwecke nicht besonders geeignet.

Für $x_{n+1} = f(x_n)$ heißt

$$y_{n+1} = \left.\frac{df}{dx}\right|_{x=x_n} \cdot y_n$$

die Variationsgleichung. Der (maximale) Ljapunow-Exponent ergibt sich damit für einen Anfangswert $y_0 \neq 0$ zu

$$\lambda = \lim_{n\to\infty} \frac{1}{n} \ln\left|\frac{y_n}{y_0}\right|$$

Bei numerischen Rechnungen wird man bei diesem Term etwa n in der Größenordnung von einigen Tausend wählen. Da die Variationsgleichung die Form eines Produktes hat, besteht hier die Gefahr eines numerischen Überlaufs, dem man manchmal durch logarithmische Berechnung entgehen kann.

Der Ljapunow-Exponent λ charakterisiert das Verhalten einer Funktion durch sein Vorzeichen. Für periodische Funktionen gilt $\lambda < 0$, für quasiperiodische $\lambda = 0$ und für chaotische Funktionen $\lambda > 0$. Die Umkehrung gilt nicht in jedem Fall: So hat auch die Funktion $f(x) = e^{ax}$ $(a > 0)$ einen positiven Ljapunow-Exponenten ohne chaotisch zu sein.

Das Ljapunow-Diagramm kann mit Hilfe des Turbo Pascal-Programms `ljap.pas` gefertigt werden.

```
program ljap; {exakter Ljapunow-Exponent}

{$N+}
uses crt,graph;

const T = 5000;
var j : integer;
    x,x1,y,lambda,r : double;
    Graphdriver,Graphmode : integer;
    first : boolean;

begin
GraphDriver := Detect;
Initgraph(GraphDriver,GraphMode,'\tp\bgi');
SetgraphMode(Graphmode);
cleardevice;
rectangle(0,0,639,479); { VGA-Auflösung }
line(0,120,639,120);
setcolor(14);
MoveTo(10,10);
OutText('Ljapunow-Exponent der logistischen Gleichung 2.9<r<4');

r := 2.9; first := true;
while r<= 4.0 do
```

```
  begin
  x1 := 0.61; y := 0;
  for j:=1 to T do
    begin
    x := x1;
    x1 := r*x*(1-x);
    y := y+ln(abs(r*(1-2*x)))
    end;
  lambda := y/T;
  if first then
   moveto(round(640*r-1856),round(120-120*lambda))
  else
   lineto(round(640*r-1856),round(120-120*lambda));
  first := false;
  r := r+0.00171875
  end;
repeat until keypressed;
closegraph;
TextMode(lastmode);
end.
```

2.6 Numerische Berechnung des Ljapunow-Exponenten

Eine numerisch leichter anwendbare Methode wurde von Brandstäter u.a. (Brandstäter A./Swift J./Swinney H.L:/Wolf A.: *A strange attractor in a Couette-Taylor-Experiment*, IUTAM 1984, Elsevier North Holland, 1984) bzw. von Benettin u.a. (Benettin G./Giorgilli A./Strelcyn J.M.: *Meccanica*, März (1980), 21) angegeben.

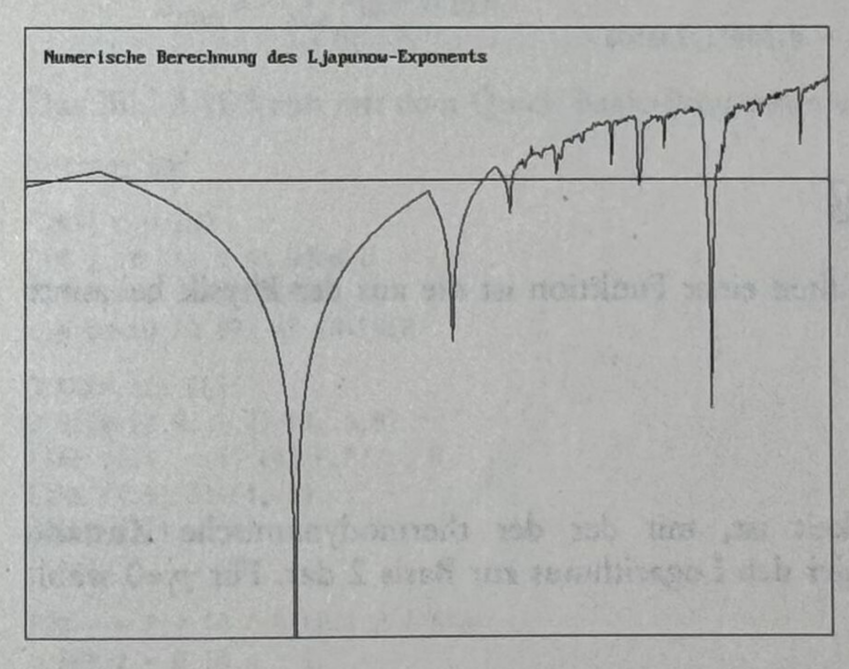


Bild 2.9 Ljapunow-Diagramm mit numerisch berechneten Exponenten.

Man berechnet zwei benachbarte Bahnen x_i und y_i, wobei die Bahn y_i mit dem Anfangswert $y_0 = x_0 + \varepsilon$ startet. Der Ljapunow-Exponent λ ist dann für große n näherungsweise gegeben durch

$$\lambda = \frac{1}{n}\sum_{i=1}^{n} \ln \frac{|y_i - x_i|}{|y_{i-1} - x_{i-1}|}$$

Das Quick Basic-Programm `brandst.bas` berechnet numerisch den (maximalen) Ljapunow-Exponenten.

```
'brandst.bas
'Berechnung des Ljapunow-Exponenten nach Brandstäter

CONST eps = .001: r = 4!
CONST T = 3000
DIM lambda, d, x, y AS DOUBLE
DIM i AS INTEGER

x = .61:lambda = 0
FOR i = 1 TO 300
  x = r * x * (1 - x)
NEXT i
FOR i = 1 TO T
  y = x + eps
  x = r * x * (1 - x)
  y = r * y * (1 - y)
  d = (y - x) / eps
  IF d <> 0 THEN lambda = lambda + LOG(ABS(d))
NEXT i
lambda = lambda / T
PRINT USING "Max.Ljapunow-Exponent = #.###"; lambda
END
```

2.7 Das Entropiemaß

Ein weiteres Maß für das Verhalten einer Funktion ist die aus der Physik bekannte Entropie

$$S = -\sum_{i=1}^{n} p_i \operatorname{ld} p_i$$

wobei p_i die Wahrscheinlichkeit ist, mit der der thermodynamische Zustand i angenommen wird; ld stellt hier den Logarithmus zur Basis 2 dar. Für p_i=0 wählt man den Grenzwert

$$\lim_{p \to \infty} p \operatorname{ld} p = 0$$

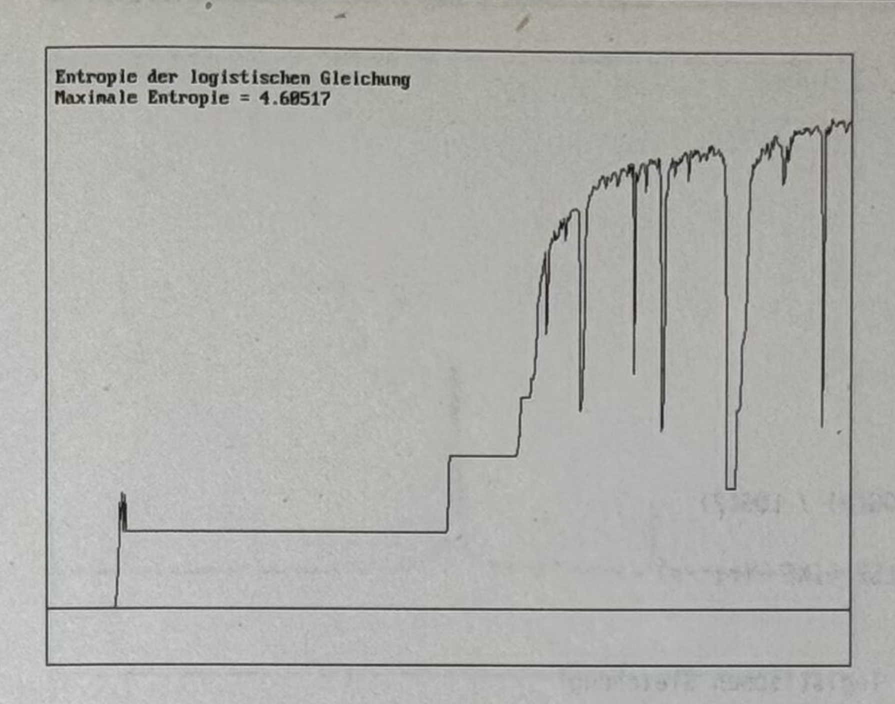


Bild 2.10 Entropie der logistischen Gleichung

Diese Wahrscheinlichkeit p_i wird ermittelt, indem man den Wertebereich der Funktion in n Teilintervalle teilt und zählt, wie oft der Funktionswert in den entsprechenden Bereich fällt.

Das Maximum der Entropie erhält man im Fall der Gleichverteilung, d.h. wenn alle Werte des Wertebereichs mit gleicher Wahrscheinlichkeit $p_i = \frac{1}{n}$ angenommen werden. Für das Entropie-Maximum ergibt sich

$$S_{\max} = -\sum_{i=1}^{n} \frac{1}{n} \operatorname{ld} \frac{1}{n} = \operatorname{ld} n$$

Das Bild 2.10 kann mit dem Quick Basic-Programm `entropy.bas` erstellt werden.

```
'entropy.bas

CONST n = 100
DIM p, r, s, x AS DOUBLE
DIM f, i, z AS INTEGER
DIM box(0 TO 99) AS INTEGER

SCREEN 12: CLS
WINDOW (2.9, -.5)-(4, 6.8)
LINE (2.9, -.5)-(4, 6.8), , B
LINE (2.9, 0)-(4, 0)
COLOR (12)
f = 1

FOR r = 2.9 TO 4 STEP 1.2 / 640
  FOR z = 0 TO n - 1
    box(z) = 0
  NEXT z
```

```
x = .61
FOR i = 1 TO 250: 'Transienten
  x = r * x * (1 - x)
NEXT i
FOR i = 1 TO 500
   x = r * x * (1 - x)
   z = INT(x * n)
   box(z) = box(z) + 1
NEXT i
s = 0
FOR z = 0 TO n - 1
   p = box(z) / 500
   IF p > 0 THEN s = s + p * LOG(p) / LOG(2)
NEXT z
IF f = 1 THEN PRESET (r, -s) ELSE LINE -(r, -s)
f = 0
NEXT r
LOCATE 2, 2: PRINT "Entropie der logistischen Gleichung"
e$ = INPUT$(1): SCREEN 0
END
```

2.8 Das Fourier-Spektrum

Ein weiteres Hilfsmittel zur Charakterisierung einer Funktion ist die Fourier-Analyse. Dabei wird die Funktion als Linearkombination von harmonischen Funktion dargestellt. Daher läßt sich eine etwa auftretende Periodenverdopplung direkt aus einer graphischen Darstellung ersehen.

Die diskrete Fourier-Transformation bestimmt eine Funktion als Linearkombination von Sinus- und Cosinus-Kurven

$$c_k = \frac{1}{n}\sum_{i=0}^{n-1} x_i \exp(-j2\pi k \frac{i}{n})$$

dabei ist die Anzahl der Funktionswerte n meist eine Zweierpotenz. Der Realteil liefert die Cosinus-Entwicklung

$$a_k = \frac{1}{n}\sum_{i=0}^{n-1} x_i \cos(2\pi k \frac{i}{n})$$

Entsprechend ergibt der Imaginärteil die Sinus-Entwicklung

$$b_k = -\frac{1}{n}\sum_{i=0}^{n-1} x_i \sin(2\pi k \frac{i}{n})$$

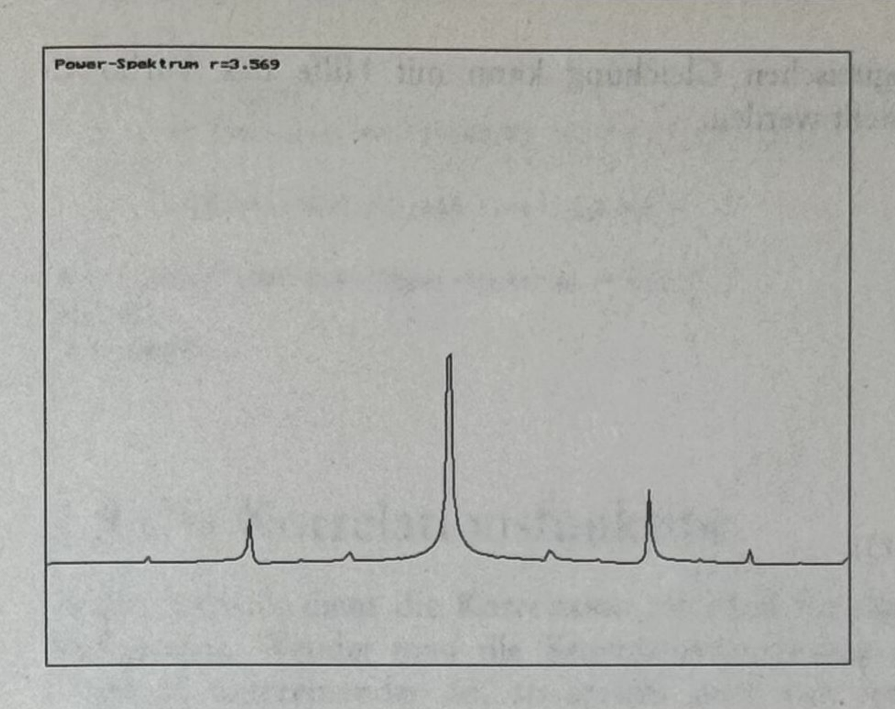


Bild 2.11 Power-Spektrum für $r = 3.569$

Der quadratische Mittelwert

$$p_k = \sqrt{a_k^2 + b_k^2}$$

heißt das *Leistungs-* oder *Power-Spektrum.*

Das Bild 2.11 zeigt die aufgetretenen Periodenverdopplungen der logistischen Funktion bei $r = 3.569$. Bild 2.12 demonstriert eine Vielzahl von Perioden beim chaotischen Verhalten, die hier bei $r = 3.85$ auftritt.

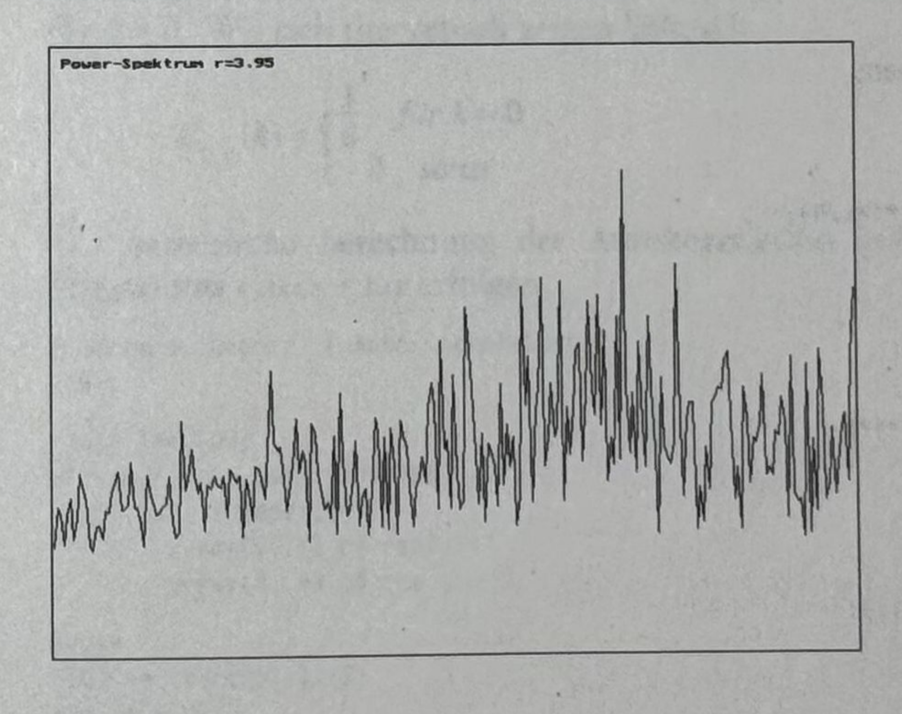


Bild 2.12 Power-Spektrum für r=3.85

Das Power-Spektrum der logistischen Gleichung kann mit Hilfe das Turbo C-Programms `fourier.c` dargestellt werden.

```
/* fourier.c */

#include <stdio.h>
#include <graphics.h>
#include <stdlib.h>
#include <math.h>
#include <conio.h>

const double PI = 3.14152965;
const double r = 3.569;
const int NN = 512;
double x[512],a[512],b[512],p[512];

void main(void)
{
int i,k,N,gdriver,gmode;
double pmax,sum,mean;
detectgraph(&gdriver,&gmode);
initgraph(&gdriver,&gmode,"\\bc\\bgi");
rectangle(0,0,639,479);
setcolor(12);

N = NN/2-1;
x[0] = 0.61;
for (i=1,mean=0; i<=NN; i++)
   {
   x[i] = r*x[i-1]*(1.-x[i-1]);
   mean += x[i];
   }
mean *= 1./NN;
for (i=0; i<=NN; i++) x[i] -= mean;
for (i=0; i<=N; i++)
   {
   for (k=0,sum=0; k<2*N; k++)
      sum += x[k]*cos((double)PI*i*k/N);
   a[i] = sum/N;
   }
for (i=0; i<N; i++)
   {
   for (k=0,sum=0; k<2*N; k++)
      sum += x[k]*sin((double)PI*i*k/N);
   b[i] = sum/N;
   }
for (i=0,pmax=0; i<=N; i++)
   {
   p[i] = sqrt(a[i]*a[i]+b[i]*b[i]);
   if (p[i]>pmax) pmax = p[i];
   }
for (i=1; i<N; i++)
```

```
    {
    p[i] *= 1./pmax;
    if (i==1) moveto((int)(i*640/N),440-(int)(800*p[i]));
    else
    lineto((int)(i*640./N),440-(int)(800.*p[i]));
    }
moveto(10,10);outtext("Power-Spektrum r=3.569");
getch();
closegraph();
}
```

2.9 Die Korrelationsfunktion

In der Statistik dient die Korrelation als Maß für die lineare Abhängigkeit zweier Meßgrößen. Wendet man die Korrelationsrechnung auf Werte einer Größe oder Funktion untereinander an, so spricht man von Autokorrelation. Die Autokorrelationsfunktion ist definiert durch

$$C_{xx}(k) = \lim_{n\to\infty} \frac{1}{n} \sum_{i=0}^{n} (x_i - \bar{x}_i)(x_{i+k} - \bar{x}_i)$$

und hängt vom Anfangswert x_0 ab. Für die logistische Gleichung gilt $\bar{x}_i = \frac{1}{2}$ (für fast alle Anfangswerte), wie man es für $r = 4$ wegen der Symmetrie der Dichtefunktion erwartet.

Kennzeichen einer chaotischen Funktion ist das Verschwinden der Autokorrelation für $k > 0$. Wie sich theoretisch zeigen läßt, gilt

$$C_{xx}(k) = \begin{cases} \frac{1}{8} & \text{für } k = 0 \\ 0 & \text{sonst} \end{cases}$$

Die numerische Berechnung der Autokorrelation kann mittels des Turbo-Pascal-Programms `autocorr.pas` erfolgen.

```
program autocorr;  { Auto-Korrelation }
{$N+}

const T=10000;
var corr,mean,sum : double;
    i,j,k: integer;
    x : array[0..T] of real;
    c : array[0..9] of real;

begin
x[0] := (sqrt(5)-1)/2;
sum := x[0];
```

```
for i:=1 to T do
  begin
  x[i] := 4*x[i-1]*(1-x[i-1]);
  sum := sum + x[i]
  end;
mean := sum/(T+1);
for i:=0 to 9 do
begin
  corr := 0;
  k := T -10;
  for j:=0 to k do
     corr := corr+(x[j]-mean)*(x[j+i]-mean);
  c[i] := corr/k;
end;
writeln('Koeffizienten der Korrelationsfunktion');
for i:=0 to 9 do writeln(i:6,c[i]:10:4);
end.
```

Das Programm liefert die Werte

k	$C_{xx}(k)$
0	0.1241
1	-0.0008
2	-0.0006
3	0.0004
4	-0.0001
5	0.0018
6	-0.0001
7	0.0014

2.10 Aufgaben

Übung 2.10.1

Zeigen Sie, daß die logistische Gleichung $f(x) = 4x(1-x)$ durch die Transformation

$$x \to \sin^2 x$$

in die Bernoulli-Gleichung (englisch *Bernoulli shift*)

$$f(x) = 2x \bmod 1$$

übergeht. Zeigen Sie, daß der Startwert $x_0 = \frac{1}{7}$ einen Zyklus der Periode 3 liefert!

Übung 2.10.2

Bestimmen Sie numerisch den Ljapunow-Exponenten für den Zufallszahlen-Generator $x_{n+1} = (x+\pi)^5 \bmod 1$ zum Startwert $x_0 = \frac{\pi}{4}$.

Übung 2.10.3

Bestimmen Sie numerisch den Ljapunow-Exponenten für den Zufallszahlen-Generator $x_{n+1} = e^{x_n+\pi} \bmod 1$ zum Startwert $x_0 = \frac{\pi}{4}$.

Übung 2.10.4

Erzeugen Sie das Bifurkations- und Ljapunow-Diagramm der Sinus-Gleichung (englisch *sine map*) $x_{n+1} = r \sin \pi x$. (Programme *sinus1,sinus2*)

Übung 2.10.5

Betrachtet werde die Zeltdach-Funktion (englisch *tent map*)

$$f(x) = \begin{cases} 2ax & \text{für } 0 < x \le \frac{1}{2} \\ 2a(1-x) & \text{für } \frac{1}{2} < x < 1 \end{cases}$$

mit dem Parameter $0 < a < 1$.

a) Bestimmen Sie den max. Ljapunow-Exponenten (Ergebnis $\lambda = \ln(2a)$).
b) Finden Sie einen Zyklus der Periode 3.
c) Erstellen Sie das Bifurkations-Diagramm (Programm *tent.bas*).
d) Zeigen Sie, daß die logistische Gleichung für $r = 4$ bei der Transformation

$$x \to \frac{2}{\pi} \arcsin \sqrt{x}$$

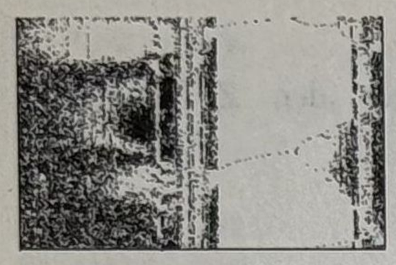
in die Zeltdachfunktion übergeht.

Übung 2.10.6

Erstellen Sie ein Ljapunow-Diagramm (Programm *ljapsin.bas*) für die Gleichung

$$x_{n+1} = b \sin^2(x_n + r); b = 2.5$$

wobei der Parameter r abwechselnd die Werte A und B haben soll. A, B durchlaufen dabei das Intervall $[0;2\pi]$. Der numerisch berechnete Ljapunow-Exponent der Iteration soll an den Koordinaten $(A|B)$ des Bildschirms geplottet werden. Dies liefert Farbtafel 2.



3 Kreisgleichung (Circle Map)

Die von V. Arnold angegebene Kreisgleichung ist gegeben durch

$$x_{n+1} = x_n + 2\pi\,\Omega + K\cos(2\pi x_n) \bmod 2\pi$$

(*Small Denominators for Mappings of the Circumference Into Itself*, AMS Transl. Series 2, 46 (1965)). Sie kann umgeformt werden zu

$$x_{n+1} = x_n + \Omega + \frac{K}{2\pi}\cos(2\pi x_n) \bmod 1$$

Die Kreisgleichung kann als Gleichung eines periodisch angeregten Oszillators verstanden werden. Der Parameter Ω stellt das ungestörte Frequenzverhältnis, der Parameter K die Kopplungskonstante der nichtlinearen Störung dar. Das Bild 3.1 entsteht durch Plotten von 5000 Funktionswerten in Abhängigkeit von dem Parameter K mit $0 < K < 1$. Dabei werden die Funktionswerte nach oben angetragen, die K-Achse liegt waagrecht.

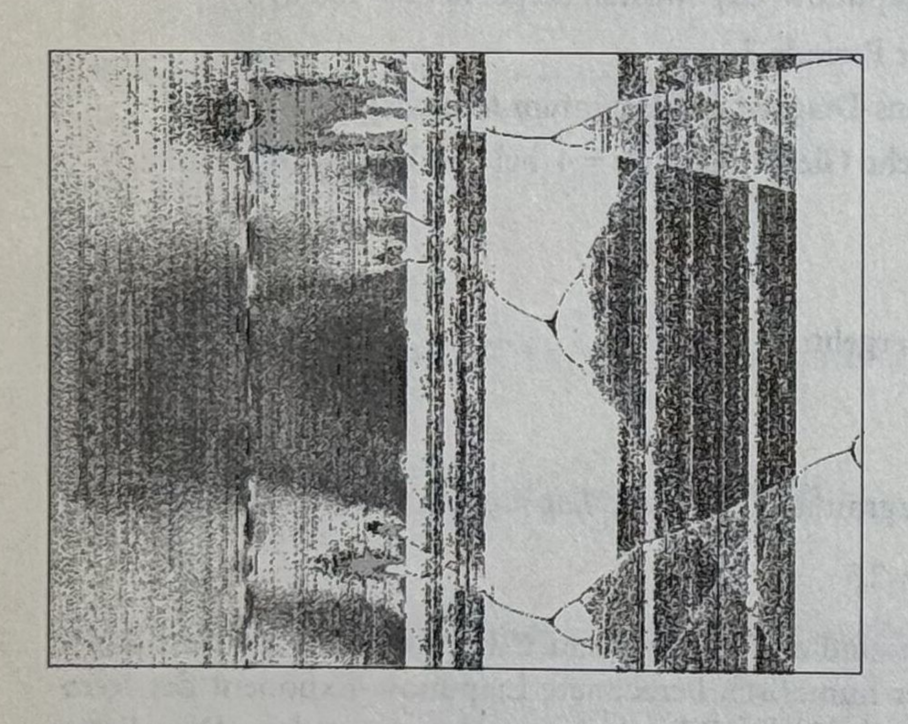


Bild 3.1 Bifurkations-Diagramm für $0 < K < 1$

Die Abbildung kann mit dem Quick Basic-Programm circ.bas erstellt werden.

```
'circ.bas
'Bifurkationsdiagramm der Circle Map

CONST pi = 3.141592653#:twopi = 2 * pi
CONST omega = .45
DIM i AS INTEGER
DIM k, x, x1 AS SINGLE
DIM h(480)
DIM col(1 TO 15) AS INTEGER

SCREEN 12: CLS
WINDOW (.2, 0)-(2, 479)

FOR i = 1 TO 14: READ col(i): NEXT i
FOR k = .2 TO 2 STEP 1.8 / 640
   FOR i = 0 TO 479: h(i) = 0: NEXT i
   x = .4
   FOR i = 0 TO 300
      x1 = x + omega + k / twopi * COS(twopi * x)
      x = x1 - INT(x1)
      IF i > 100 THEN h(x * 480) = 1
   NEXT i
   FOR i = 0 TO 5000
      x1 = x + omega + k / twopi * COS(twopi * x)
      x = x1 - INT(x1)
      z = INT(480 * x)
      h(z) = h(z) + 1
   NEXT i
   FOR i = 0 TO 479
   IF h(i) > 15 THEN h(i) = 15
   IF h(i) > 0 THEN PSET (k, i), col(h(i))
   NEXT i
NEXT k
e$ = INPUT$(1): SCREEN 0
END
DATA 15,7,8,11,3,10,2,9,1,14,6,12,4,13,5
```

3.1 Fixpunkte

Aus der Fixpunktgleichung $x_{n+1} = x_n$ erhält man die Bedingung

$$0 = \Omega + \frac{K}{2\pi}\cos(2\pi\ x)$$

oder $$x = \frac{1}{2\pi}\arccos\frac{-2\pi\ \Omega}{K}$$

Für die Stabilität der Fixpunkte ergibt sich aus der Ableitung die Bedingung

$$\left|1 - K\sin(2\pi x)\right| < 1$$

bzw. $\quad -1 < 1 - K\sin(2\pi\ x) < 1$

Die linke Ungleichung kennzeichnet hier die Periodenverdopplung. Die rechte Ungleichung bedeutet die Tangenten-Bifurkation, die für kleine K ein Übergang zur Quasi-Periodizität ist. Für kleine K ist dies sicher erfüllt. Löst man die obige Gleichung für die Fixpunkte nach K auf, so erhält man die Hyperbel

$$K = \sqrt{4 + (2\pi\Omega)^2}$$

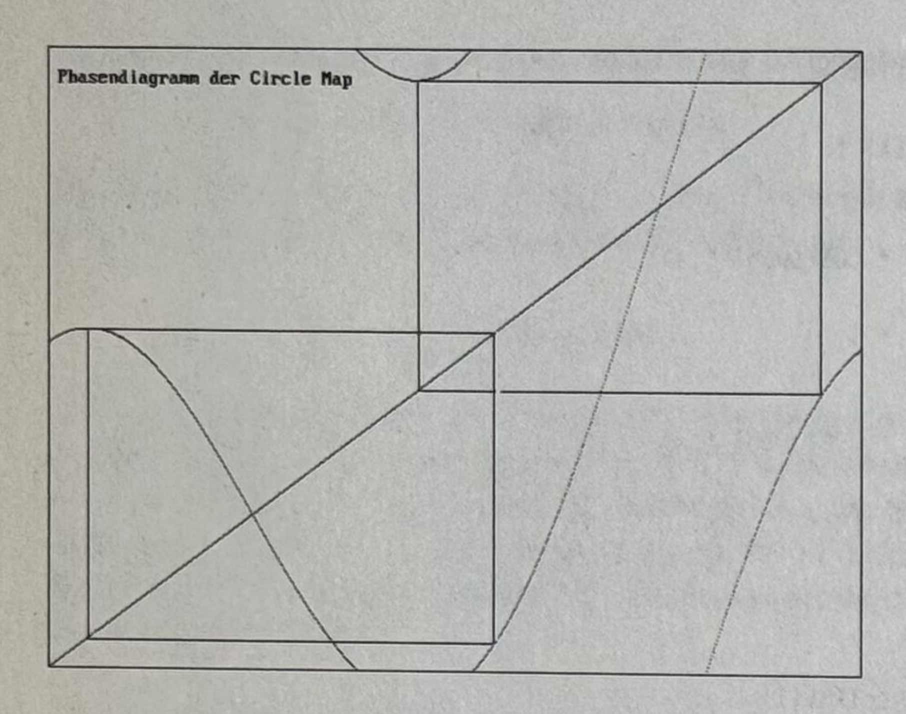


Bild 3.2 2 superstabile Bahnen für K=3.3,$\Omega = 0$

3.2 Der Fall K nahe Null

Setzt man für K=0 und $\Omega = \frac{p}{q}$, so vereinfacht sich die Kreisgleichung zu

$$x_{n+1} = x_n + \frac{p}{q} \bmod 1$$

oder beim Startwert x_0 nach q-maliger Iteration

$$x_q = x_0 + p \bmod 1 = x_0$$

d.h. man erhält einen Zyklus der Länge q unabhängig vom Startwert x_0. Für irrationales Ω schließt sich der Zyklus nicht exakt und man erhält ein quasiperiodisches Verhalten. Für $K \neq 0$ definiert man die Windungszahl

$$W = \lim_{n \to \infty} \frac{x_n - x_o}{n}$$

dabei wird auf die Restbildung mod 1 verzichtet. Tritt wegen der Periodizität keine Änderung der Windungszahl auf, so spricht man von Phasensperre (englisch *phase locking*). Das Bild 3.2 zeigt die Phasenabbildung mit zwei stabilen Zyklen.

Trägt man in einem Diagramm die Windungszahl W gegen Ω auf , so erhält man eine monoton steigende Kurve mit einigen Plateaus, die Teufelstreppe (englisch *devil's staircase*) genannt wird (Bild 3.3).

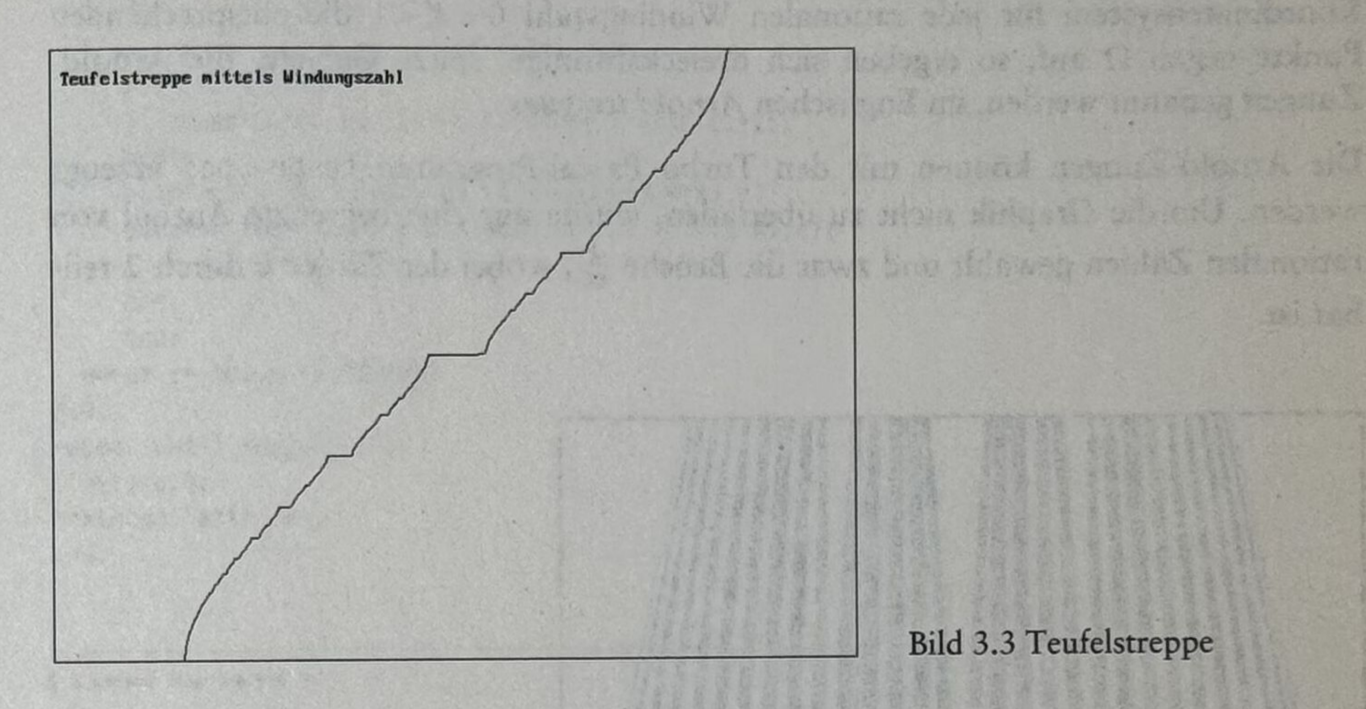


Bild 3.3 Teufelstreppe

Das Quick Basic-Programm `devil.bas` erstellt die Graphik zur Teufelstreppe.

```
'devil.bas
'Teufelstreppe der Circle Map

CONST PI = 3.141592653#: p2 = 2 * PI
CONST K = 1
DIM x, w AS SINGLE
DIM i AS INTEGER

SCREEN 12: CLS
WINDOW (0, 0)-(1, 1)
LINE (0, 0)-(1, 1), , B
PRESET (0, 0), 12

FOR omega = 0 TO 1 STEP .0015625
  x = PI / 4
  FOR i = 1 TO 4000
     x = x + omega + K * COS(p2 * x) / p2
  NEXT i
  w = (x - x0) / 4000!
  LINE -(omega, w), 12
NEXT omega
```

```
LOCATE 2, 2: PRINT "Teufelstreppe mittels Windungszahl"
e$ = INPUT$(1): SCREEN 0
END
```

Die Ableitung der Funktion ist für $0 < K < 1$ stets positiv

$$f'(x) = 1 - K\sin(2\pi\ x)$$

Dies bedeutet, daß alle Fixpunkte für diese K-Werte stabil sind. Plottet man in einem Koordinatensystem für jede rationalen Windingszahl $0 < K < 1$ die entsprechenden Punkte gegen Ω auf, so ergeben sich dreiecksförmige, spitze Gebiete, die Arnold-Zungen genannt werden, im Englischen *Arnold tongues*.

Die Arnold-Zungen können mit den Turbo Pascal-Programm `tongue.pas` erzeugt werden. Um die Graphik nicht zu überladen, wurde nur eine begrenzte Anzahl von rationalen Zahlen gewählt und zwar die Brüche $\frac{k}{60}$, wobei der Zähler k durch 2 teilbar ist.

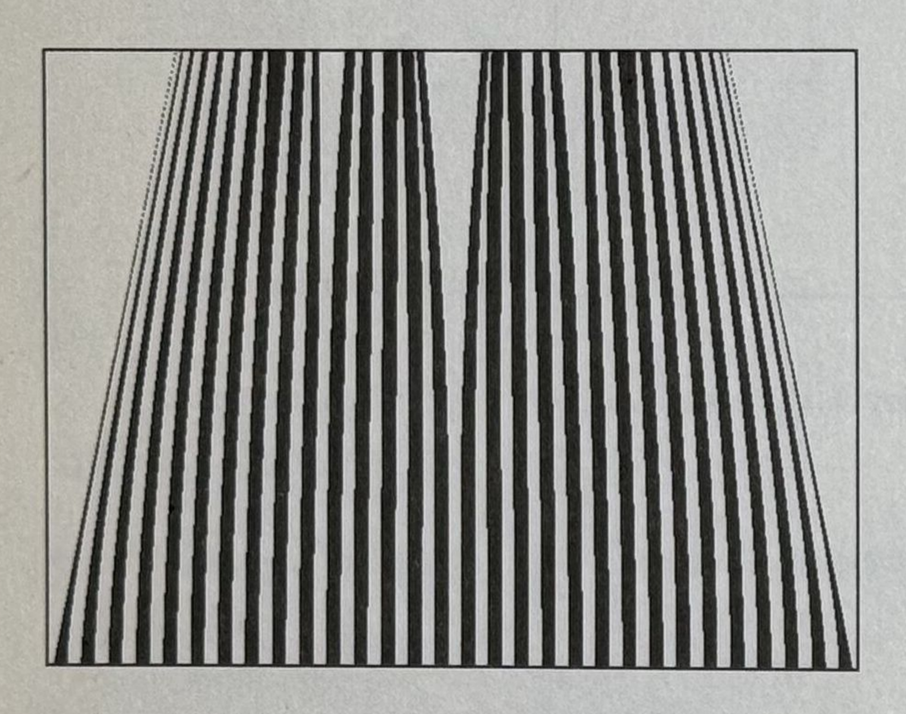


Bild 3.4 Arnold-Zungen

```
program devil;
uses crt,graph;

const p2 = 2*pi;
var i,c,z : integer;
    K,x,omega,w : real;
Graphdriver,Graphmode:integer;

begin
GraphDriver := Detect;
Initgraph(GraphDriver,GraphMode,'\tp\bgi');
SetgraphMode(Graphmode);
```

```
cleardevice;

omega := 0;
while omega<= 0.5 do
  begin
  K := 0; z:=0;
  while K<=1.0 do
     begin
     x := 0.1;
     for i:=1 to 30 do
         x := x + omega - K*sin(p2*x)/p2;
     w := (x-x0)/30;
     if round(60*w) mod 2 <>0 then c:= 1 else c:=15;
     begin
     putpixel(round(639*omega),479-round(479*K),c);
     putpixel(639-round(639*omega),479-round(479*K),c);
     K := K + 0.00208333
     end
     end;
  omega := omega +0.0015625
end;
repeat until keypressed;
closegraph;
TextMode(lastmode);
end.
```

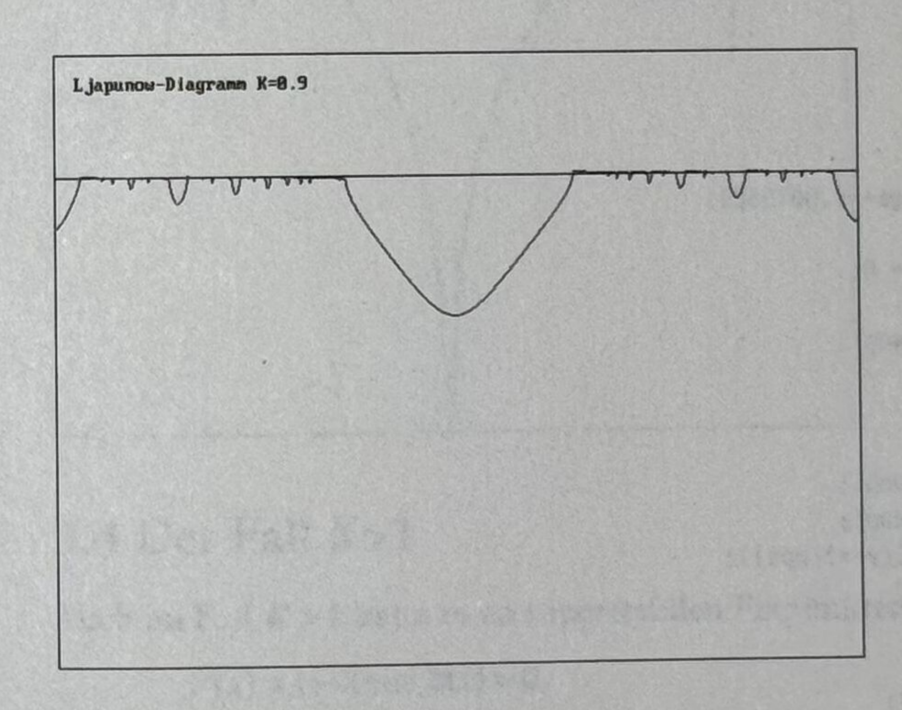


Bild 3.5 Ljapunow-Diagramm für K=0.9 als Funktion von $-\frac{1}{2} \leq \Omega \leq \frac{1}{2}$

Für infinitesimal kleines, positives K gilt näherungsweise

$$\lambda = \sum_i \ln\left|f(x_i)\right| \approx \sum_i -K \sin(2\pi\; x_i)$$

Dies bedeutet, daß der Ljapunow-Exponent negativ bzw. Null ist. Dies zeigt erneut, daß es sich hier um periodische bzw. quasiperiodische Bahnen handelt (vgl. Bild 3.5)

Der Ljapunow-Exponent der Kreisgleichung kann numerisch nach Brandstäter berechnet werden. Dies führt das Turbo C-Programm `ljap.c` durch.

```
/* ljap.c */
/* Numerische Berechnung des Ljapunow-Exponenten */

#include <stdio.h>
#include <math.h>
#include <graphics.h>
#include <conio.h>

typedef int BOOL;
const double eps = 0.015;
const double K = 0.9;
const double twopi = 6.283185307;
const int T = 5000;

void main(void)
{
int i,p,q;
BOOL first = 1;
double lambda,omega,x,y;
int gdriver,gmode;
gdriver = 9; gmode = VGAHI;
initgraph(&gdriver,&gmode," ");
rectangle(0,0,639,479);
line(0,96,639,96);
setcolor(14);

for (omega=-0.5; omega<=0.5; omega+=0.0015625)
  {
  x = (sqrt(5.)-1.)*0.5; lambda = 0;
  for (i=1; i<=400; i++)
     x += omega+K*cos(twopi*x)/twopi;
  y = x + eps;
  for (i=1; i<=T; i++)
     {
     x += omega+K*cos(twopi*x)/twopi;
     y += omega+K*cos(twopi*y)/twopi;
     if (x!=y) lambda +=log(fabs((y-x)/eps));
     y = x + eps;
     }
  lambda *= 1./T;
  p = 320+(int) floor(640.*omega);
  q = (int) floor(48.*(2.-lambda));
  if (first) moveto(p,q); else lineto(p,q);
  first = 0;
  }
moveto(5,5);outtext("Ljapunow-Diagramm K=0.9");
```

```
do {} while(!(kbhit()));
closegraph();
return;
}
```

3.3 Der Fall K=1

Im Fall K=1 kann die Ableitung

$$f'(x) = 1 - K\sin(2\pi\ x)$$

verschwinden, was superstabile Fixpunkte bedeutet (vgl. auch Bild 3.2). Im Fall einer Periode 1, lassen sich die Positionen der Gabel-Bifurkationen angeben; sie liegen auf der Kurve

$$K = \sqrt{1 + (2\pi\ \Omega)^2}$$

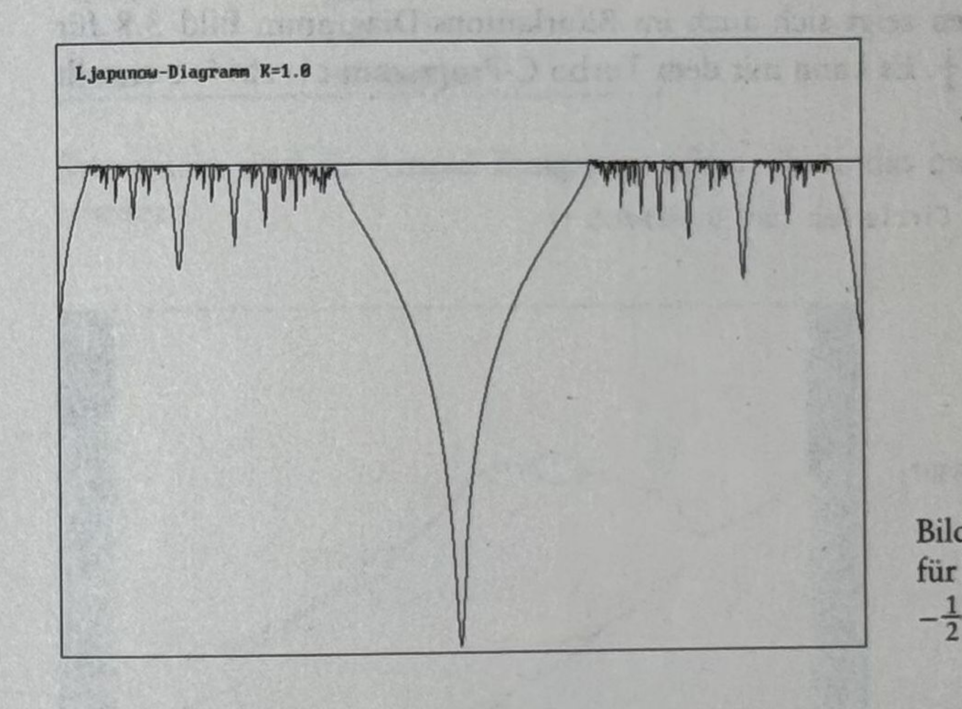


Bild 3.6 Ljapunow-Diagramm für K=1.0 als Funktion von $-\frac{1}{2} \leq \Omega \leq \frac{1}{2}$

3.4 Der Fall K>1

Auch im Fall $K > 1$ kann es zu superstabilen Fixpunkten kommen, falls die Gleichung

$$f'(x) = 1 - K\sin(2\pi x) = 0$$

erfüllt ist, andernfalls kommt es zu chaotischem Verhalten. Der Übergang läßt sich an Hand des Ljapunow-Diagramms betrachten (Bild 3.7). Bei K=2.7 zeigt sich, daß für ungefähr $|\Omega| > 0.45$ der Ljapunow-Exponent positiv wird und somit Chaos eintritt.

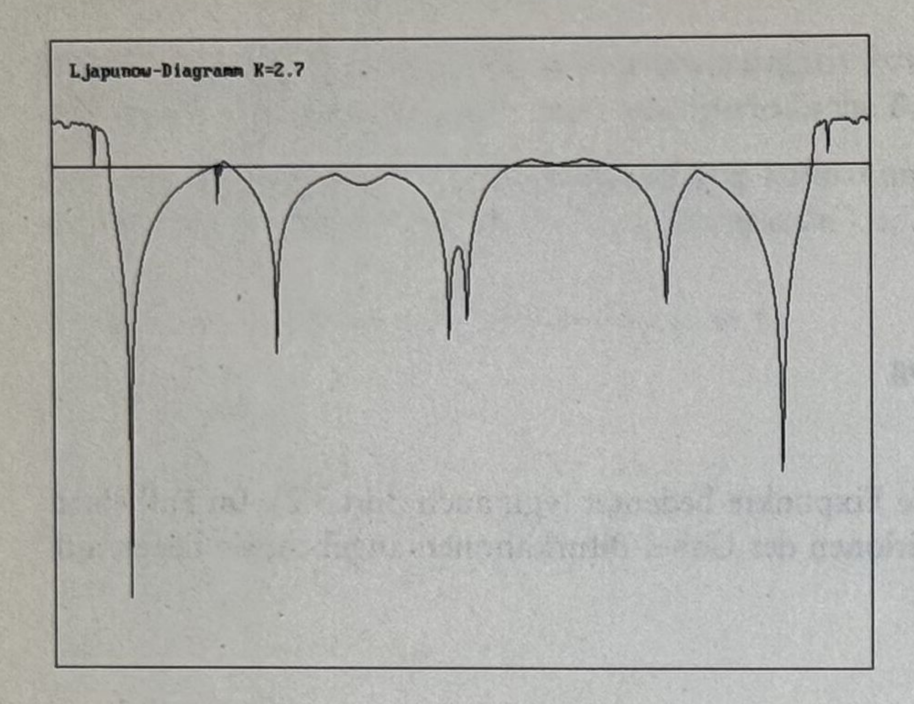


Bild 3.7 Ljapunow-Diagramm für K=2.7 als Funktion von $-\frac{1}{2} \leq \Omega \leq \frac{1}{2}$

Dieses chaotische Verhalten zeigt sich auch im Bifurkations-Diagramm Bild 3.8 für festes K=2.7 und $-\frac{1}{2} < \Omega < \frac{1}{2}$. Es kann mit dem Turbo C-Programm `circbif.c` erstellt werden.

```
/* circbif.c */
/* Bifurkations-Diagramm der Circle Map for -0.5<Ω<0.5 */

#include <stdio.h>
#include <math.h>
#include <graphics.h>
#include <conio.h>

const double K = 2.7;
const double twopi = 6.283185307;

void main(void)
{
int i,p,q;
float omega,x;
int gdriver,gmode;
gdriver = 9; gmode = VGAHI;
initgraph(&gdriver,&gmode," ");
rectangle(0,0,639,479);
setcolor(14);

for (omega=-0.5; omega<=0.5; omega+=0.0015625)
  {
  x = 0.3;
  for (i=1; i<=1200; i++)
    {
    x += omega+K*cos(twopi*x)/twopi;
    x -= floor(x);
```

```
        p = 320+(int)(floor(640.*omega));
        q = 480-(int)(floor(480.*x));
        if (i>1000) putpixel(p,q,9);
        }
    }
do {} while(!(kbhit()));
closegraph();
return;
}
```

Färbt man in einem Ω-K-Diagramm jeden Punkt gemäß dem Wert des Ljapunow-Exponenten, so erhält man Farbtafel 1 (siehe Farbtafel-Teil des Buchs). Hierbei wurde folgende Färbung gewählt

λ	Farbe
$\lambda < -2$	grün
$-2 < \lambda < 0$	blau
$\lambda = 0$	gelb
$0 < \lambda < 1$	rot
$\lambda > 1$	dunkelrosa

Sehr schön sind die Arnold-Zungen zu sehen, die in das quasiperiodische Gebiet hineinragen.

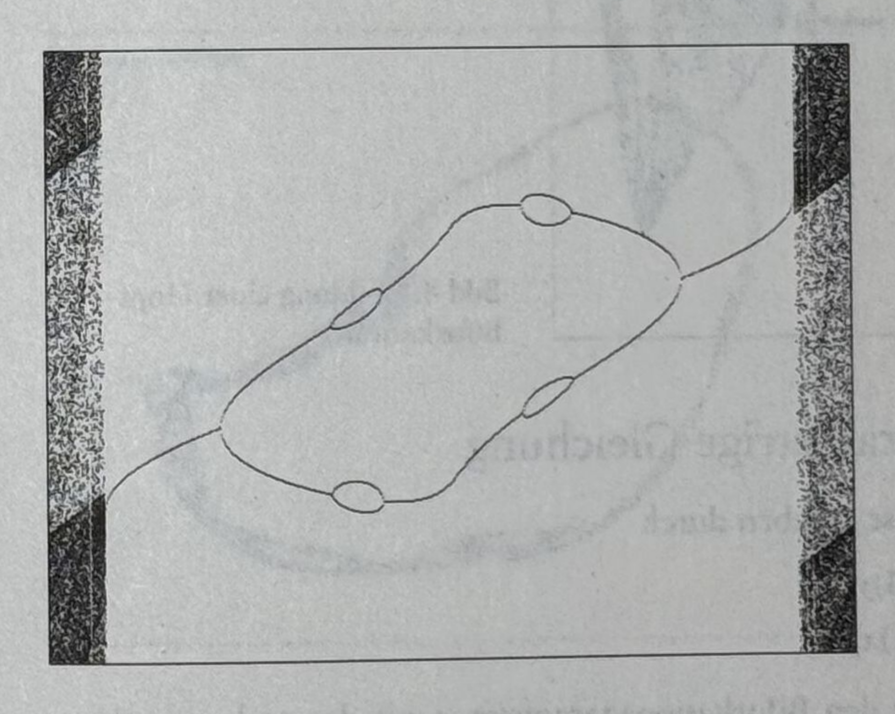


Bild 3.8 Bifurkations-Diagramm für $-\frac{1}{2} \leq \Omega \leq \frac{1}{2}$



4 Zweidimensionale logistische Gleichung

Die logistische Gleichung $f(x) = rx(1-x)$ kann auf verschiedene Arten auf zwei Dimensionen verallgemeinert werden. Dies ist in sofern interessant, da manches chaotische Verhalten, z.B. Hopf-Bifurkationen, nicht bei einem eindimensionalen System auftreten kann.

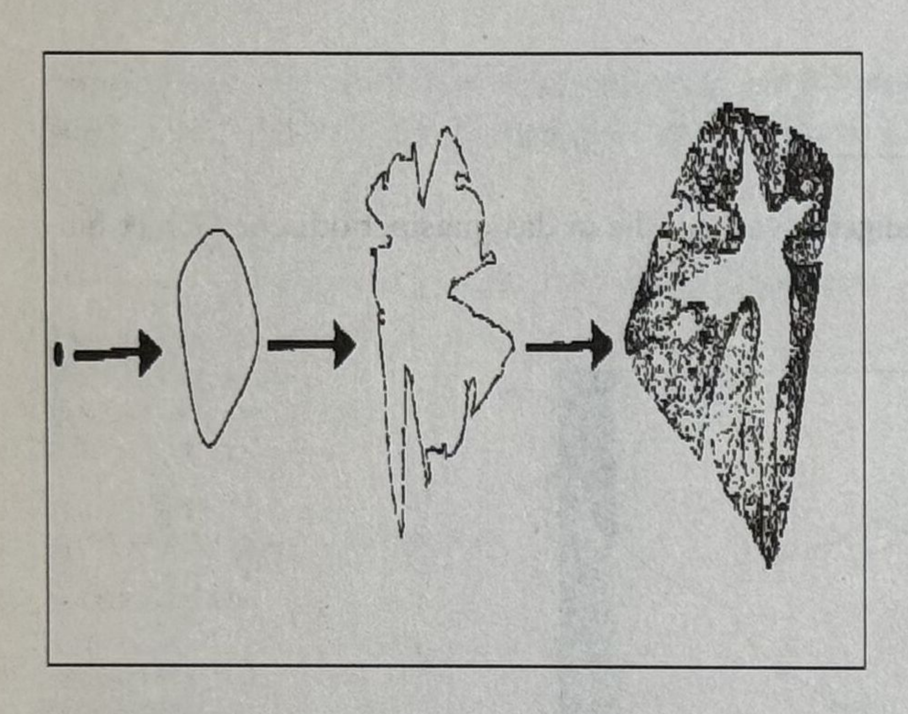


Bild 4.1 Bildung einer Hopf-Bifurkation

4.1 Gekoppelte einparametrige Gleichung

Eine erste Verallgemeinerung ist gegeben durch

$$x_{n+1} = rx_n(1-x_n) + (r-1)y_n$$
$$y_{n+1} = ry_n(1-y_n) + (r-1)x_n$$

Dabei ist jede Gleichung über den Bifurkationsparameter r mit der anderen gekoppelt. Plottet man das Phasendiagramm des Systems, so erhält man für $r = 1.667$ elliptische Bahnen, für größer werdende Parameterwerte bis $r = 1.8$ eine geschlossene Kur-

ve, die für $r = 1.815$ in drei Kurvenstücke und für $r = 1.8233$ in Einzelpunkte zerfällt. Für $r = 1.8252$ stellt sich eine Hopf-Bifurkation ein.

Das Phasendiagramm kann mit dem Quick Basic-Programm `logist2.bas` erstellt werden.

```
'logist2.bas

CONST r = 1.6667
DIM x, x1, y AS DOUBLE

SCREEN 12: CLS
WINDOW (.75, .75)-(.85, .85)

FOR j = 0 TO 6
x = j / 7: y = x
FOR i = 1 TO 2000
  x1 = r * x * (1 - x) + (r - 1) * y
  y = r * y * (1 - y) + (r - 1) * x1
  x = x1
  PSET (x, y), 1 + j
NEXT i
NEXT j
LOCATE 2, 2: PRINT "Phasenplot a=1.6667"
e$ = INPUT$(1): SCREEN 0
END
```

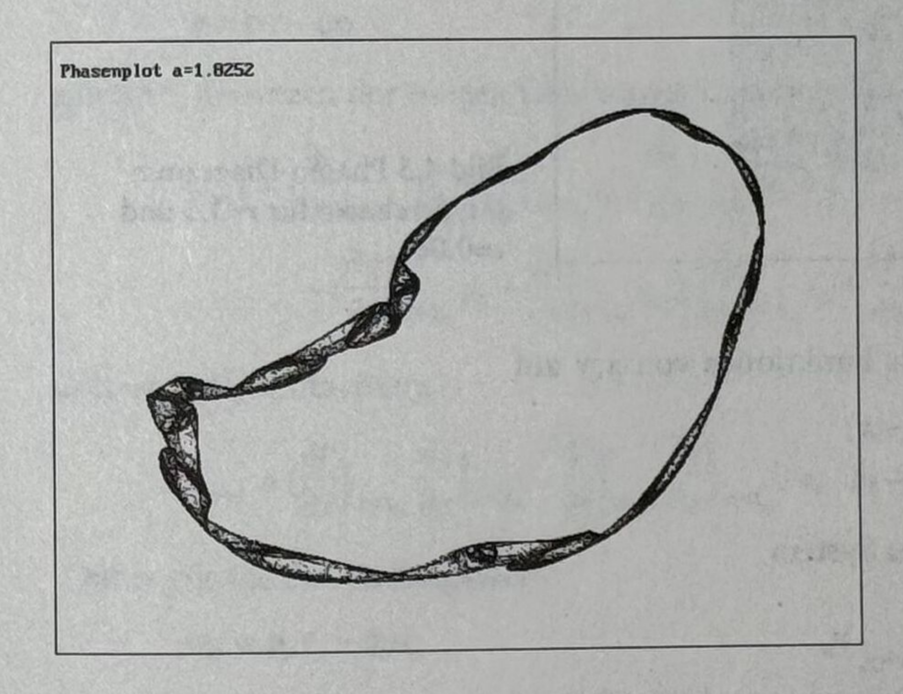


Bild 4.2 Hopf-Bifurkation für r=1.8252

4.2 Gekoppelte, zweiparametrige Gleichung

Eine zweite Verallgemeinerung erhält man, wenn man einen zweiten Parameter ε als Kopplungskonstante einführt.

$$x_{n+1} = rx_n(1-x_n)+\varepsilon(y_n - x_n)$$
$$y_{n+1} = ry_n(1-y_n)+\varepsilon(x_n - y_n)$$

Dieses System zeigt außer periodischen, quasiperiodischen und chaotischen Verhalten auch Hyperchaos. Von Hyperchaos spricht man, wenn der maximale eindimensionale Ljapunow-Exponent $\lambda^{(1)}$und der entsprechende zweidimensionale $\lambda^{(2)}$ positiv sind. Dies ist der Fall für $r = 3.6$ und ε=0.06 (siehe Bild 4.3).

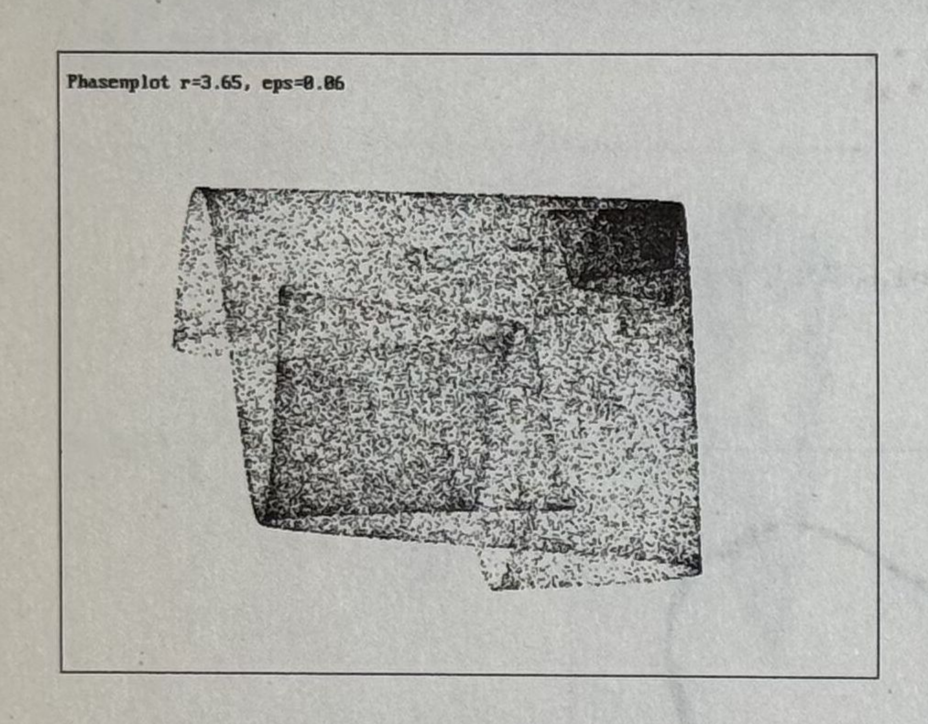


Bild 4.3 Phasen-Diagramm (Hyperchaos) für r=3.6 und ε=0.06

Faßt man die rechten Seiten als Funktionen von x, y auf

$$f(x,y) = rx(1-x)+\varepsilon(y-x)$$
$$g(x,y) = ry(1-y)+\varepsilon(x-y)$$

so liefert die Linearisierung das System

$$u_{n+1} = \left.\frac{\partial f}{\partial x}\right|_{x=x_n} u_n + \left.\frac{\partial f}{\partial y}\right|_{y=y_n} v_n$$
$$v_{n+1} = \left.\frac{\partial g}{\partial x}\right|_{x=x_n} u_n + \left.\frac{\partial g}{\partial y}\right|_{y=y_n} v_n$$

Diese läßt sich formal schreiben als Produkt mit der Jacobi-Matrix

$$\mathbf{J} = \begin{pmatrix} \frac{\partial f}{\partial x} & \frac{\partial f}{\partial y} \\ \frac{\partial g}{\partial x} & \frac{\partial g}{\partial y} \end{pmatrix}$$

Einsetzen der Funktionen f und g liefert die Variationsgleichung

$$p_{n+1} = (r(1-2x)-\varepsilon)p_n + \varepsilon q_n$$
$$q_{n+1} = \varepsilon p_n + (r(1-2x)-\varepsilon)q_n$$

Ist $(\mathbf{u},\mathbf{v})$ eine Lösung der Variationsgleichung, so gilt

$$u_{n+1} = (r(1-2x)-\varepsilon)u_n + \varepsilon v_n$$
$$v_{n+1} = \varepsilon u_n + (r(1-2x)-\varepsilon)v_n$$

Den eindimensionalen Ljapunow-Exponenten $\lambda^{(1)}$ erhält man damit aus

$$\lambda^{(1)} = \lim_{n\to\infty} \frac{1}{n} \ln(|u_n| + |v_n|)$$

Hier wurde für den Vektor $(\mathbf{u},\mathbf{v})$ die Betragsnorm verwendet, prinzipiell läßt sich jede Vektornorm $\|.\!.\|$ anwenden. Zur Berechnung des zweidimensionalen Ljapunow-Exponenten wird das Graßmann-Produkt der Vektoren $(\mathbf{p},\mathbf{q})$ und $(\mathbf{u},\mathbf{v})$

$$\mathbf{w} = \mathbf{pv} - \mathbf{qu}$$

gebildet. Einsetzen der obigen Gleichungen liefert

$$\mathbf{w}_{n+1} = \left(\frac{\partial f}{\partial x}\Big|_{x=x_n} p_n + \frac{\partial f}{\partial y}\Big|_{y=y_n} q_n\right)\left(\frac{\partial g}{\partial x}\Big|_{x=x_n} u_n + \frac{\partial g}{\partial y}\Big|_{y=y_n} v_n\right)$$
$$-\left(\frac{\partial g}{\partial x}\Big|_{x=x_n} p_n + \frac{\partial g}{\partial y}\Big|_{y=y_n} q_n\right)\left(\frac{\partial f}{\partial x}\Big|_{x=x_n} u_n + \frac{\partial f}{\partial y}\Big|_{y=y_n} v_n\right)$$

und nach Vereinfachung

$$w_{n+1} = \left(\frac{\partial f}{\partial x}\Big|_{x=x_n} \frac{\partial g}{\partial y}\Big|_{y=y_n} - \frac{\partial f}{\partial y}\Big|_{y=y_n} \frac{\partial g}{\partial x}\Big|_{x=x_n}\right) w_n$$

Dabei gilt für den Anfangswert

$$w_0 = p_o v_o - q_o u_o$$

Einsetzen der Ableitungen liefert schließlich die gesuchte Iteration

$$w_{n+1} = (r^2(1-2x)(1-2y) - 2r\varepsilon(1-x-y))w_n$$

Der zweidimensionale Ljapunow-Exponent ist damit bestimmt durch

$$\lambda^{(2)} = \lim_{n\to\infty} \frac{1}{n} \ln|w_n|$$

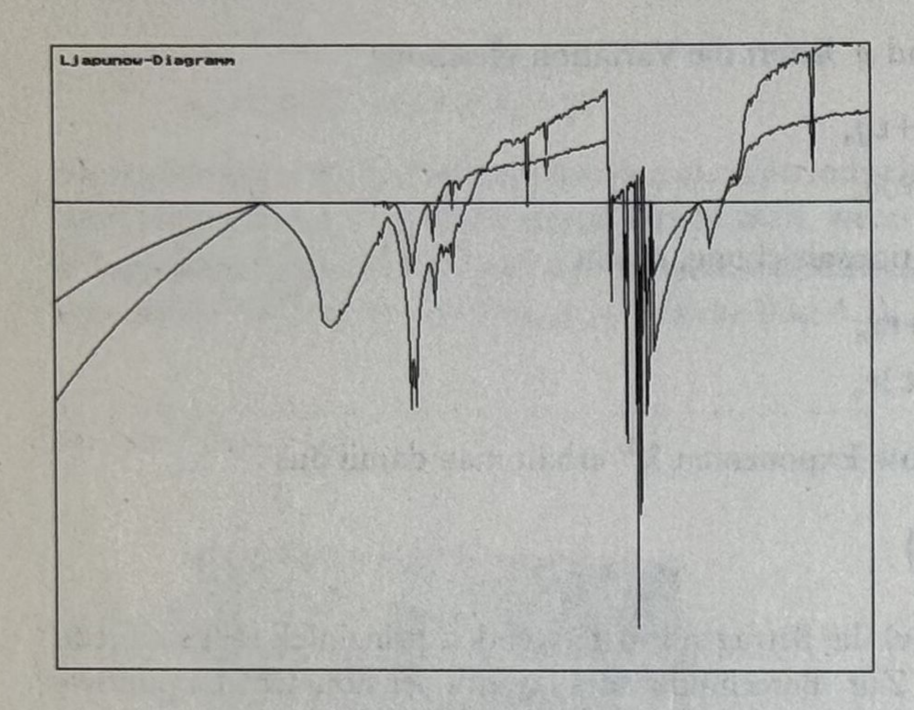


Bild 4.4 Ljapunow-Diagramm im Bereich 3.2 < r <3.7

Für die Parameterwerte $\varepsilon = 0.06$, $r = 3.70$ tritt Hyperchaos auf. Die beiden maximalen Ljapunow-Exponenten sind hier $\lambda^{(1)} = 0.38$ bzw. $\lambda^{(2)} = 0.69$. Sie werden durch das Turbo Pascal-Programm `lljap2.pas` berechnet und geplottet.

```
program lja2;
{Ljapunow-Exponenten der gekoppelten logistischen Gleichung }
{$N+}
uses crt,graph;

const eps = 0.06; t = 5000;
    rmin = 3.2; rmax = 3.7;

var i,p,p1,q,q1,s,s1:integer;
    r : real;
    first : boolean;
    lambda1,lambda2,x,x1,y,y1,u,u1,v,v1,w : extended;
    Graphdriver,Graphmode:integer;

begin
randomize;
GraphDriver := Detect;
Initgraph(GraphDriver,GraphMode,'\tp\bgi');
SetgraphMode(Graphmode);
cleardevice;
rectangle(0,0,639,479);
line(0,120,639,120);
```

```
r := rmin; first := true;
while r<=rmax do
begin
  x := 0.2; y := 0.4;
  u := 0.5; v := 0.5; w := 0;
  for i:=1 to t do
     begin
     x1 := r*x*(1-x)+eps*(y-x);
     y1 := r*y*(1-y)+eps*(x-y);
     u1 := (r-2*r*x-eps)*u+eps*v;
     v1 := eps*u+(r-2*r*y-eps)*v;
     w := w+ln(abs(r*r*(1-2*x)*(1-2*y)-2*r*eps*(1-x-y)));
     x := x1; y:= y1; u := u1; v := v1;
     end;
  lambda1 := ln(abs(u)+abs(v))/t;
  lambda2 := w/t;
  p := round((r-rmin)*640/(rmax-rmin));
  q := 480-round((lambda1+2)*180);
  s := 480-round((lambda2+2)*180);
  if not first then
    begin
    setcolor(12);line(p1,q1,p,q);
    setcolor(14);line(p1,s1,p,s);
    end;
  p1 := p; q1 := q; s1 := s;
  r := r+(rmax-rmin)/640;
  first := false;
end;
moveto(8,8);outtext('Ljapunow-Diagramm');
repeat until keypressed;
textmode(lastmode)
end.
```

4.3 Kaneko I

K. Kaneko publizierte 1986 folgendes System (*Collapse of Tori and the Genesis of Chaos in Dissipative Systems*, Singapore: World Scientific, 1986)

$$x_{n+1} = 1 - ax_n^2 + d(y_n - x_n)$$
$$y_{n+1} = 1 - ay_n^2 + d(x_n - y_n)$$

das durch die Transformation

$$a = r\left(\frac{r}{4} - \frac{1}{2}\right); \quad d = \varepsilon; \quad x = \frac{x - \frac{1}{2}}{\frac{r}{4} - \frac{1}{2}}; \quad y = \frac{y - \frac{1}{2}}{\frac{r}{4} - \frac{1}{2}}$$

direkt in die gekoppelte, zweiparametrige logistische Gleichung übergeht.

Für festgehaltenes $a=0.3$ und $d=1.75$ erhält man eine Grenzkurve, die man als Oszillation auf einem Torus deuten kann. Steigert man d bis 1.86, kommt es zur Phasensperre. Ab $d=1.94$ erscheinen Chaosbänder auf dem Torus. Bei $d=2.04$ zerbrechen die Bänder und sind bei $d=2.16$ in Hyperchaos zerfallen.

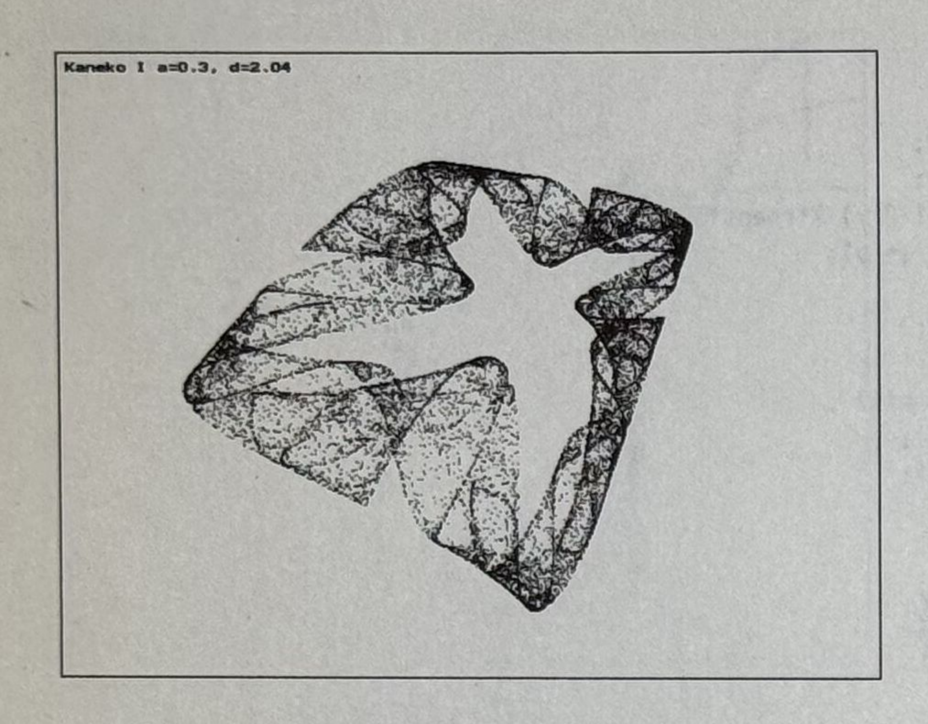


Bild 4.5 Kaneko I mit a=0.3, d=2.04

Das System kann mittels des Turbo C-Programms `kaneko.c` dargestellt werden.

```
/* kaneko.c */
/* Phasenplot des Kaneko-Systems */

#include <stdio.h>
#include <graphics.h>
#include <math.h>
#include <conio.h>

const double a = 0.3,d = 2.04;

void main(void)
{
int i;
double x,y,x1;
int gdriver,gmode;
gdriver = 9; gmode = VGAHI; /* VGA-Modus 640 x 480 */
initgraph(&gdriver,&gmode," ");
rectangle(0,0,639,479);
x = y = 0.1;
for (i=0; i<=30000; i++)
   {
   x1 = a*x+(1-a)*(1-d*y*y);
   y = x;
   x = x1;
```

```
    putpixel(240+floor(320.*x),320-floor(250.*y),1+i/3000);
    }
moveto(8,8);outtext("Kaneko I a=0.3, d=2.04");
do {} while(!(kbhit()));
closegraph();
return;
}
```

4.4 Kaneko II

Die Gleichung Kaneko I wurde vom selben Autor linearisiert zu

$$x_{n+1} = ax_n + (1-a)(1-d|y_n|)$$
$$y_{n+1} = x_n$$

Der Fixpunkt

$$(x|y) = \left(\frac{1}{1+d}\middle|\frac{1}{1+d}\right)$$

ist nicht stabil, so daß schnell chaotisches Verhalten auftritt. Im Gegensatz zur Gleichung Kaneko I ist hier die Jacobi-Matrix konstant. Mit den Gleichungen

$$f(x,y) = ax + (1-a)(1-d|y|)$$
$$g(x,y) = x$$

ergibt die Jacobi-Derminante

$$\det \mathbf{J} = \begin{vmatrix} a & (a-1)d \\ 1 & 0 \end{vmatrix} = (1-a)d$$

Wegen der Konstanz der Determinante ist die Summe der eindimensionalen Ljapunow-Exponenten gegeben durch

$$\lambda_1 + \lambda_2 = \ln|\det \mathbf{J}| = \ln|(1-a)d|$$

Das charakteristische Polynom erhält man aus der Determinante

$$\det(\mathbf{J} - \mu\mathbf{E}) = \begin{vmatrix} a-\mu & (a-1)d \\ 1 & -\mu \end{vmatrix} = 0$$

Die Lösung der quadratischen Gleichung liefert

$$\mu = \frac{1}{2}(a \pm \sqrt{a^2 - 4(1-a)d})$$

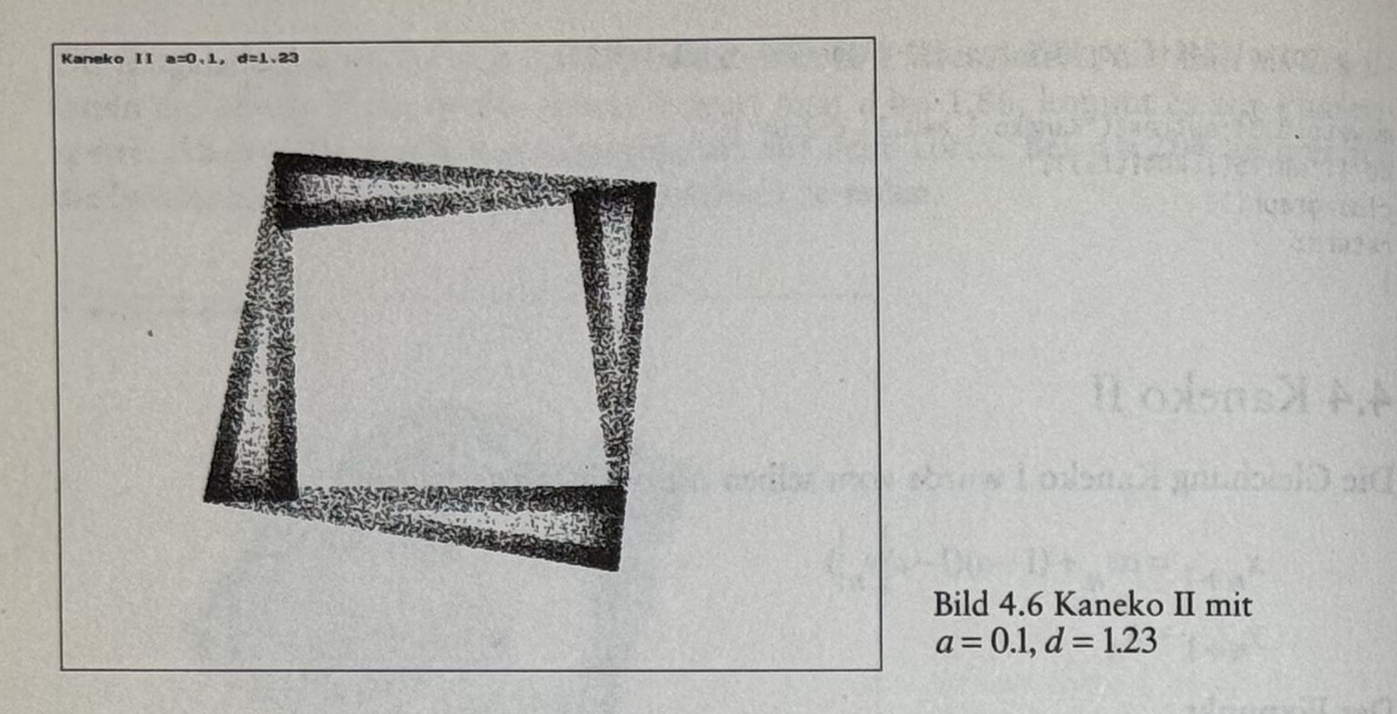


Bild 4.6 Kaneko II mit $a = 0.1, d = 1.23$

Dies zeigt, daß die Eigenwerte μ für $a = 0.1; d = 1.23$ konjugiert komplex werden. Da konjugiert-komplexe Zahlen betragsgleich sind, folgt

$$\lambda_1 = \frac{1}{2}\ln(\det(\mathbf{J})) = \frac{1}{2}\ln|(1-a)d|$$

Für die angegebenen Parameterwerte ergibt der Ljapunow-Exponent

$$\lambda_1 = 0.102 > 0$$

und zeigt damit chaotisches Verhalten.

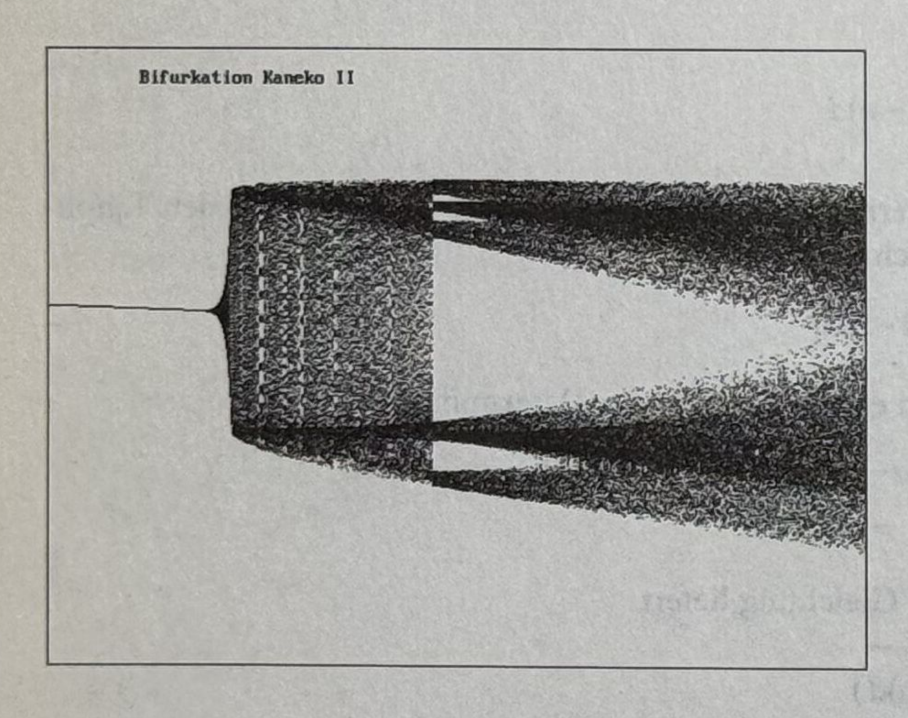
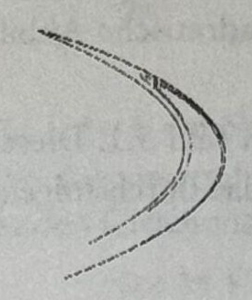


Bild 4.7 Bifurkationsdiagramm von Kaneko II für $1 < d < 1.7$



5 Hénon-Abbildung

Die 1976 von Hénon (M. Hénon: *A twodimensional mapping with a strange attractor*, Comm. Math. Phys., 50, 69-78) angegebene Abbildung (englisch *Hénon map*) lautet

$$x_{n+1} = 1 + y_n - ax_n^2$$
$$y_{n+1} = bx_n$$

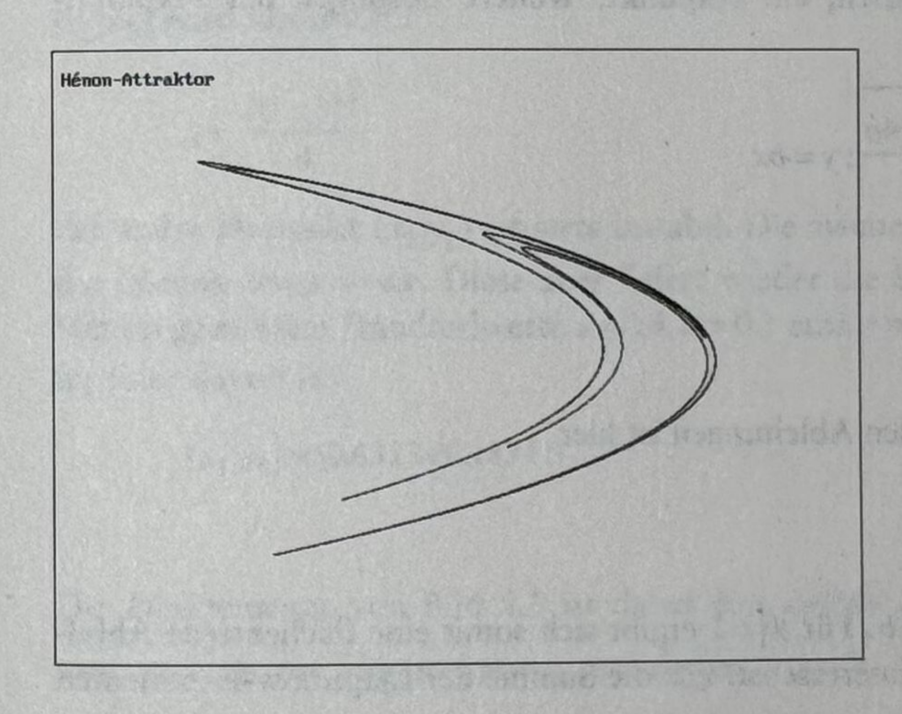


Bild 5.1 Hénon-Attraktor

Nach dem Vorbild des Hufeisen-Modells von Smale ist die Hénon-Abbildung konstruiert durch Verkettung folgender Funktionen

$T': x' = x; y' = y + 1 - ax^2$ (*Faltung in parabelähnliche Form*)

$T'': x'' = bx'; y'' = y'$ (*Stauchung in x-Richtung*)

$T''': x''' = y''; y''' = x''$ (*Spiegelung an Winkelhalbierenden*)

Nachträglich erkannte Hénon, daß die von ihm – zunächst willkürlich – gewählte quadratische Form mit zwei Parametern bereits die allgemeinste quadratische Abbildung mit konstanter Jacobi-Determinante darstellt.

Für die Parameterwerte $a = 1.4$, $b = 0.3$ fand Hénon den Attraktor von Bild 5.1. Dieser Attraktor ist beschränkt, da er innerhalb des Vierecks $ABCD$ liegt, das durch folgende Eckpunkte bestimmt ist

$$(x_A|y_A) = (-1.33|-0.42)$$

$$(x_B|y_B) = (1.32|0.133)$$

$$(x_C|y_C) = (1.245|-0.14)$$

$$(x_D|y_D) = (-1.06|-0.5)$$

5.1 Fixpunkte der Hénon-Gleichung

Der Ursprung ist, wie ersichtlich, ein Fixpunkt. Weitere Lösungen der Fixpunkt-Gleichung sind

$$x = \frac{-1 + b \pm \sqrt{(1-b)^2 + 4a}}{2a}; \; y = bx$$

Diese sind reell für

$$a \geq -\frac{(1-b)^2}{4}$$

Die Jacobi-Matrix der partiellen Ableitungen ist hier

$$\mathbf{J} = \begin{pmatrix} -2ax & 1 \\ b & 0 \end{pmatrix}$$

mit der Determinante $\det \mathbf{J} = -b$. Für $|b| = 1$ ergibt sich somit eine flächentreue Abbildung, die aber hier nicht von Interesse ist. Für die Summe der Ljapunow-Exponenten folgt somit

$$\lambda_1 + \lambda_2 = \ln|b|$$

Für die Eigenwertgleichung oder charakteristisches Polynom folgt

$$\det\begin{vmatrix} -2ax-\mu & 1 \\ b & -\mu \end{vmatrix} = 0 \text{ oder } \mu^2 + 2ax\mu - b = 0$$

Da $|\mu| = 1$ die Stabilitätsgrenze der Fixpunkte ist, macht man den Ansatz $\mu = e^{i\varphi}$. Die Eigenwertgleichung zerfällt nach Trennung der Real- und Imaginärteile in die beiden Gleichungen

$$\cos 2\varphi + 2ax\cos\varphi - b = 0$$
$$\sin 2\varphi + 2ax\sin\varphi = 0$$

Die letztere Gleichung hat die Lösungen

$$\sin\varphi = 0 \wedge \cos\varphi = \pm 1$$

Einsetzen in die erste Gleichung zeigt, daß es nur für

$$a = -\frac{(1-b)^2}{4}$$

eine Lösung gibt für den Fall, daß die beiden Fixpunkte zusammenfallen. Das negative Vorzeichen ergibt $\mu = -1$ und liefert eine Periodenverdopplung. Der Fixpunkt $(x_1|x_2)$ wird instabil für

$$a = \frac{3(1-b)^2}{4}$$

der andre Fixpunkt $(x_2|y_2)$ ist stets instabil. Die zweite Gleichung hat ebenfalls noch die Lösung $\cos\varphi = -ax$. Diese aber liefert wieder die Bedingung $b = -1$. Für die von Hénon gewählten Standardwerte $a = 1.4, b = 0.3$ erhält man zwei nichtstabile Fixpunkte; einer davon ist

$$\left(x_1|y_1\right) = (0.63134|0.18941)$$

Die Punktemenge von Bild 5.1 ist daher eine nichtstabile Mannigfaltigkeit. Interessant ist, daß das Hénon-System umkehrbar ist; d.h. aus $\left(x_{n+1}|y_{n+1}\right)$ läßt sich die Iteration eindeutig zurückrechnen auf $\left(x_n|y_n\right)$. Die so erhaltene stabile Punktemenge ist in Bild 5.2 ersichtlich. Sie kann mit dem Programm Quick Basic-Programm `henon2.bas` erhalten werden.

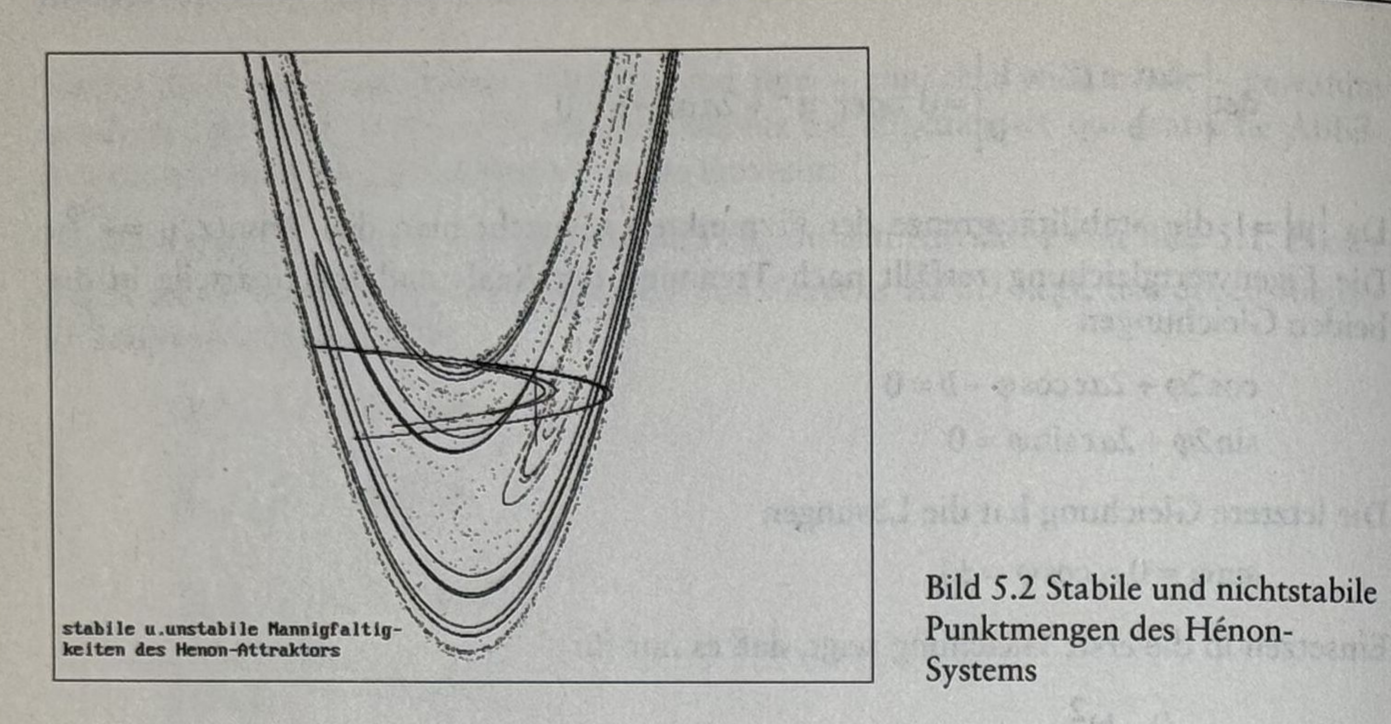


Bild 5.2 Stabile und nichtstabile Punktmengen des Hénon-Systems

```
'henon2.bas

CONST a = 1.4: b = .3
DIM x, x1, y AS DOUBLE
DIM i AS LONG

SCREEN 12: CLS
WINDOW (-3.5, -2.4)-(3.5, 2.8)
LINE (-3.5, -2.4)-(3.5, 2.8), , B
RANDOMIZE TIMER
x = .1: y = .1

FOR i = 1 TO 20000
  x1 = 1 + y - a * x * x
  y = b * x
  x = x1
  IF i > 20 THEN PSET (x, y), 9
NEXT i

FOR i = 1 TO 100000
  IF RND < .5 THEN
      x = .63 * RND: y = .19
  ELSE
      x = .63: y = .19 * RND
  END IF
  DO
    x1 = y / b
    y = x - 1 + a * x1 * x1
    x = x1
    PSET (x, y), 14
  LOOP WHILE (ABS(x) < 2) AND (ABS(y) < 3)
NEXT i
```

```
LOCATE 28, 2: PRINT "stabile u.unstabile Mannigfaltig-"
LOCATE 29, 2: PRINT "keiten des Hénon-Attraktors";
e$ = INPUT$(1): SCREEN 0
END
```

5.2 Berechnung der Ljapunow-Exponenten

Aufstellen der Variationsgleichungen liefert das System

$$u_{n+1} = -2ax_n u_n + v_n$$
$$v_{n+1} = bu_n$$

Die Berechnung der Ljapunow-Exponenten mittels des Pascal-Programms `henljap.pas` liefert für die Parameterwerte $a = 1.4$; $b = 0.3$ den maximalen Ljapunow-Exponenten

$$\lambda_1 = 0.42$$

Damit ergibt sich aus der Jacobi-Determinante wegen $\lambda_1 + \lambda_2 = \ln|b| = -1.20$

$$\lambda_2 = -1.62$$

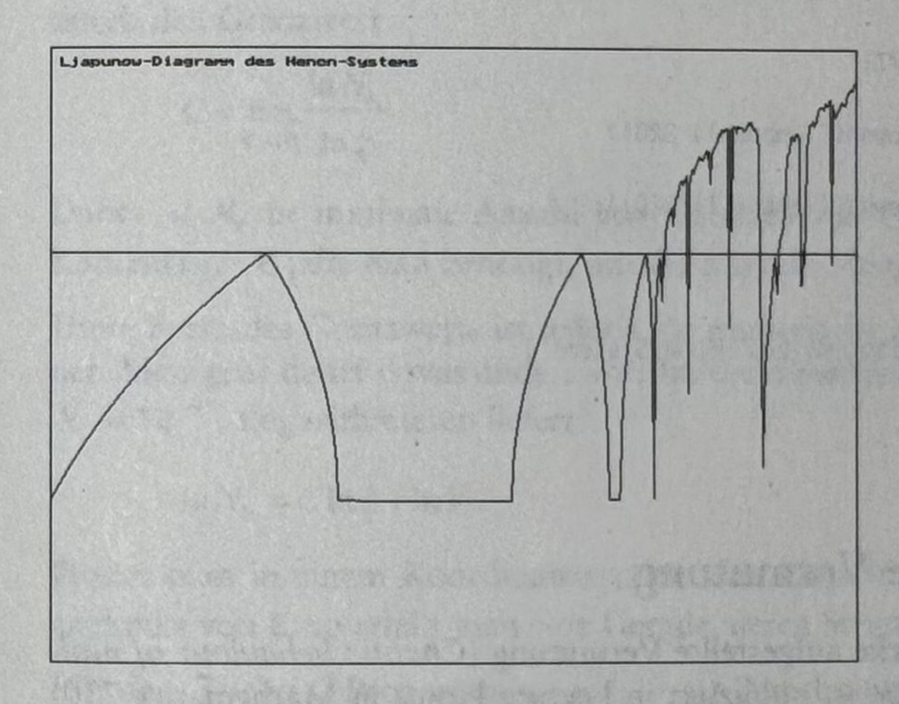


Bild 5.3 Ljapunow-Diagramm der Hénon-Abbildung

```
program henljap;
{ Ljapunow-Diagramm der Hénon-Gleichung }
{$N+}

uses crt,graph;
const b = 0.3; T = 5000;
var  a,x,xl,y,yl,u,ul,v,vl: extended;
     lambda:double;
```

```
    i : integer;
    first : boolean;
    Graphdriver,Graphmode:integer;

begin
GraphDriver := Detect;
Initgraph(GraphDriver,GraphMode,'\tp\bgi');
SetgraphMode(Graphmode);
cleardevice;
rectangle(0,0,639,479);
line(0,160,640,160);
setcolor(14);

a := 0; first := true;
while a<=1.4 do
  begin
  x := 0.1; y := 0.3; u := 0.1; v := 0.3;
  for i := 1 to T do
     begin
     x1 := 1+y-a*x*x;
     y1 := b*x;
     u1 := -2*a*x*u+v;
     v1 := b*u;
     x := x1; y := y1; u := u1; v := v1
     end;
   lambda := ln(abs(u) + abs(v))/T;
   if first then
      moveto(round(a*427),480-round((lambda+1)*320))
   else
      lineto(round(a*457),480-round((lambda+1)*320));
   a := a+0.0021875;
   first := false
   end;
moveto(8,8); outtext('Ljapunow-Diagramm des Hénon-Systems');
repeat until keypressed;
textmode(lastmode)
end.
```

5.3 Die Kaplan-Yorke Vermutung

Die 1978 von Kaplan und Yorke aufgestellte Vermutung (*Chaotic behaviour of multidimensional difference equations*, publiziert in Lecture Notes in Mathematics 730) erlaubt es, die fraktale Dimension eines Attraktors anhand der Ljapunow-Exponenten abzuschätzen. Ist j die Anzahl der positiven Exponenten und werden die Exponenten monoton fallend numeriert, so gilt für die Kaplan-Yorke-Dimension, manchmal auch Ljapunow-Dimension genannt,

$$D = j + \frac{\lambda_1 + \lambda_2 + \ldots \lambda_j}{|\lambda_{j+1}|}$$

dabei ist die natürliche Zahl j gekennzeichnet durch die obengenannten Bedingungen

$$\sum_{i=1}^{j} \lambda_i > 0 \wedge \sum_{i=1}^{j+1} \lambda_i < 0$$

Für den Hénon-Atraktor ergibt sich hieraus mit $j = 1$

$$D = 1 + \frac{\lambda_1}{|\lambda_2|} = 1.26$$

Dies ist auch der Wert, der sich in der Literatur findet (vgl. Mandelbrot [17], Seite 210).

5.4 Die Kapazität

Die Kapazität oder Box-Counting Dimension einer fraktalen Menge ist definiert durch den Grenzwert

$$C = \lim_{\varepsilon \to 0} \frac{\ln N_\varepsilon}{\ln \frac{1}{\varepsilon}}$$

Dabei ist N_ε die minimale Anzahl von Gebieten wie Quadrate, Würfel usw. mit der Kantenlänge ε, die man benötigt, um die fraktale Menge zu überdecken.

Diese Form des Grenzwerts ist jedoch für numerische Zwecke nicht besonders geeignet. Man geht daher etwas anders vor. Im dreidimensionalen Fall gilt näherungsweise $N_\varepsilon = V\varepsilon^{-C}$. Logarithmieren liefert

$$\ln N_\varepsilon = C \ln \tfrac{1}{\varepsilon} + \ln V$$

Plottet man in einem Koordinatensystem den Logarithmus von N_ε gegen den Logarithmus von ε so erhält man eine Gerade, deren Steigung die gesuchte Kapazität ist.

Mit dem Turbo C-Programm `capacity.c` zur Berechnung der Kapazität erhält man bei den Parameterwerten $a = 1.4; b = 0.3$ für den Hénon-Attraktor die Tabelle

ε	N_ε
0.008	3143
0.004	8380
0.002	19512

Lineare Regression liefert für diese Punkte die Geradensteigung $C = 1.31$. In der Literatur findet sich hier für die Box-Counting-Dimension der Wert $D = 1.21$. Siehe dazu: *Estimating the fractal dimensions and entropies of strange attractors* von P. Grassberger (Wuppertal) im Sammelband [13].

```
/* capacity.c */
/* Box-Dimension des Hénon-Attraktors */

#include <stdio.h>
#include <stdlib.h>
#include <math.h>

#define N 2048
#define XDIM 110
#define YDIM 366
const double eps = 0.002;
float capacity(float *x, float *y, unsigned char box[YDIM][XDIM]);

void main(void)
{
float x[N],y[N];
float cap;
int j;
unsigned char box[YDIM][XDIM];

x[0] = 1.161094; y[0] = -0.09541356;
for (j=0; j<N-1; j++)
   {
   x[j+1] = 1.+y[j]-1.4*x[j]*x[j];
   y[j+1] = 0.3*x[j];
   }
cap = capacity(x,y,box);
printf("Capacity = %f\n",cap);
}

float capacity(float *x, float *y,unsigned char box[YDIM][XDIM])
{
unsigned int i,j,k,Nx,Ny,Neps=0;
float mx,my,xmin,xmax,ymin,ymax;

xmin = xmax = x[0];
ymin = ymax = y[0];
for (i=1; i<N; i++)
   {
   if (x[i]<xmin) xmin=x[i];
   else if (x[i]>xmax) xmax=x[i];
   if (y[i]<ymin) ymin=y[i];
   else if (y[i]>ymax) ymax=y[i];
   }
Nx = (unsigned) ((xmax-xmin)/eps +1.);
Ny = (unsigned) ((ymax-ymin)/eps +1.);
mx = ((float) Nx-1.)/(xmax-xmin);
my = ((float) Ny-1.)/(ymax-ymin);
```

```
for (i=0; i<Ny; i++)
for (j=0; j<Nx; j++) box[i][j]=0;
for (i=0; i<N; i++)
  {
  k = (unsigned) floor(mx*(x[i]-xmin)+0.5);
  j = (unsigned) floor(my*(y[i]-ymin)+0.5);
  box[j][k] = 1;
  }
for (i=0; i<Ny; i++)
for (j=0; j<Nx; j++) Neps += (unsigned) box[i][j];
return (float)(log(Neps)/log(1./eps));
}
```

Das Programm liefert nur Näherungswerte für die Kapazität, da ε hier nicht beliebig klein gemacht werden kann. Sind die Werte N_ε = `Neps` gesucht, so kann man diese aus der Funktion `capacity` ausdrucken lassen.

5.5 Das Korrelationsintegral

Ein andere fraktale Dimension liefert das Korrelationsintegral, definiert durch

$$C(r) = \lim_{n\to\infty} \frac{1}{n^2} \sum_{i,j=1;\, i\neq j}^{n} H(r - |x_i - x_j|)$$

dabei ist $H(x)$ die *Heavyside*-Funktion. Das Korrelations-Integral verhält sich für kleine r näherungsweise gemäß dem Potenzgesetz

$$C(r) \propto r^D$$

Der Exponent D ist die gesuchte Korrelationsdimension. Das angegebene Turbo C-Programm zum Korrelations-Integral liefert für die Parameterwerte $a = 1.4; b = 0.3$ die Tabelle

r	$C(r)$
6	0.1231
7	0.1450
8	0.1687
9	0.1926
10	0.2178
11	0.2427
12	0.2670

Die lineare Regression liefert für die Logarithmen der Tabellenwerte den Wert $D = 1.12$ für die Korrelationsdimension.

Im allgemeinen kann nicht erwartet werden, daß die fraktalen Dimensionen, die nach Kaplan-Yorke bzw. mittels Kapazität und Korrelation berechnet werden, exakt übereinstimmen. Die Hoffnung der Mathematiker, hier ein einheitliches Dimensionsmaß zu finden, haben sich zerschlagen. Beim Hénon-Attraktor streuen diese Werte stärker; die fraktale Dimension liegt daher zwischen 1.21 und 1.32.

Ein Verfahren zur Berechnung des Korrelationsintegral liefert das Turbo C-Programm `corrint.c`.

```
/* corrint.c */
/* Correlations-Integral des Hénon-Attraktors */

#include <stdio.h>
#include <stdlib.h>
#include <math.h>

const double a=1.4;
const double b=0.3;
const int T = 5000;
static float x[5000],y[5000];
static long int f[13];
double c[13];

void main(void)
{
int i,j,r;
double norm,u,v;

x[0] = y[0] = 0.;
for (i=1; i<T; i++)
   {
   x[i] = 1.+y[i-1]-a*x[i-1]*x[i-1];
   y[i] = b*x[i-1];
   }
for (i=0; i<T; i++)
for (j=0; j<i; j++)
   {
   u = x[i]-x[j];
   v = y[i]-y[j];
   norm = sqrt(u*u+v*v);
   f[(int)(30*norm)]++;
   }
for (i=1; i<=12; i++) f[i] += f[i-1];
for (r=0; r<=12; r++)
   {
   c[r] = f[r]*2./T/T;
   printf("%6d %12ld %12.4lf\n",r,f[r],c[r]);
   }
}
```

5.6 Bifurkationsdiagramm

Von den beiden Parametern des Hénon-Systems wird meist der Parameter a als Bifurkationsparameter aufgefaßt. Hält man $b = 0.3$ fest und läßt a das Intervall $0 < a < 1.4$ durchlaufen, so erhält man das Bifurkationsdiagramm gemäß Bild 5.4.

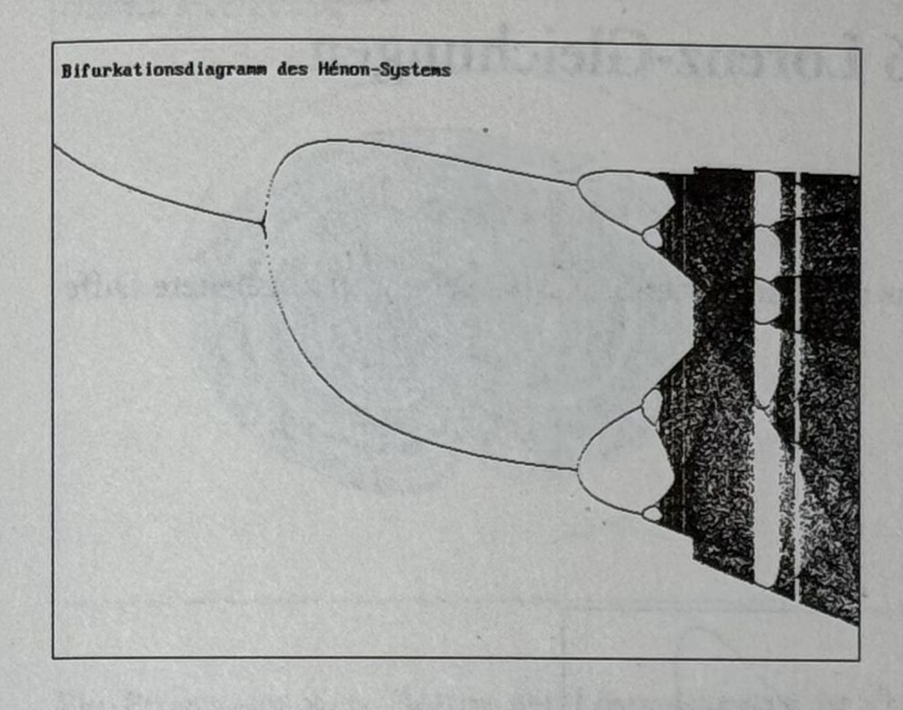


Bild 5.4 Bifurkationsdiagramm des Hénon-Systems für $0 < a < 1.4$



6 Lorenz-Gleichungen

Das von Saltzmann und Lorenz aus den Navier-Stokes-Gleichungen abgeleitete Differentialgleichungssystem lautet

$$\dot{x} = -\sigma(x-y)$$
$$\dot{y} = (r-z)x-y$$
$$\dot{z} = xy-bz$$

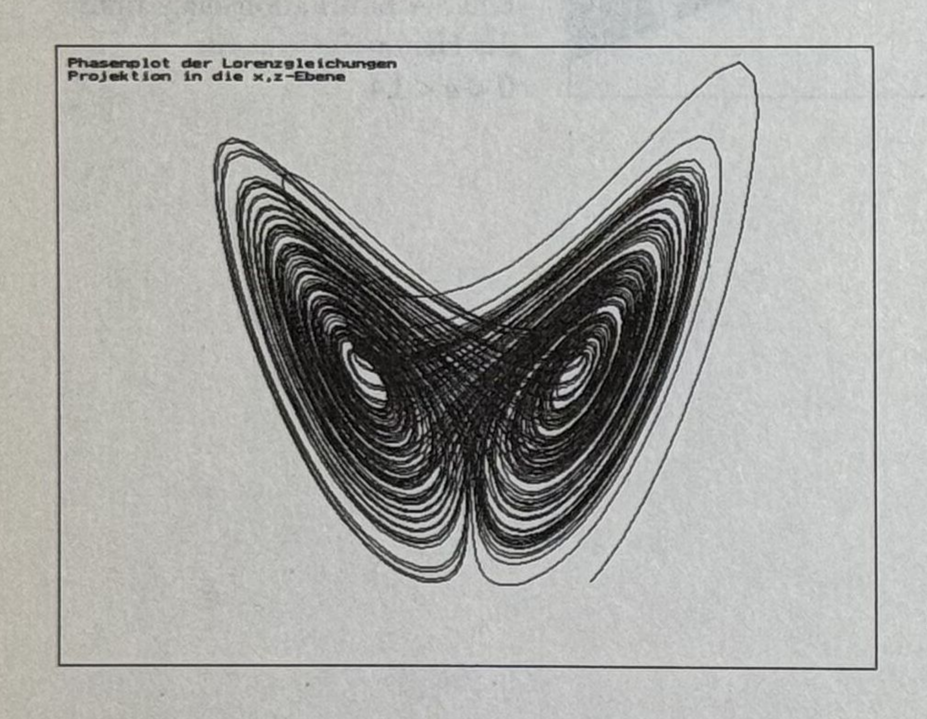


Bild 6.1 Projektion der Lorenz-Gleichungen in die x-z-Ebene

Dabei werden die Parameter, wegen ihrer physikalischen Herkunft, als positiv betrachtet: $s,b,r>0$. Die Größe x ist proportional zur Fließgeschwindigkeit, y charakterisiert die Temperaturdifferenz zwischen steigenden und fallenden Luftschichten und z ist ein Maß für die Störung des vertikalen Temperaturprofils. Wie man sieht, gehen die Gleichungen in sich selbst über, wenn man bei x und y die Vorzeichen tauscht.

Diese Symmetrieeigenschaft

$$(x, y, z) \rightarrow (-x, -y, z)$$

gilt somit auch für alle Lösungskurven.

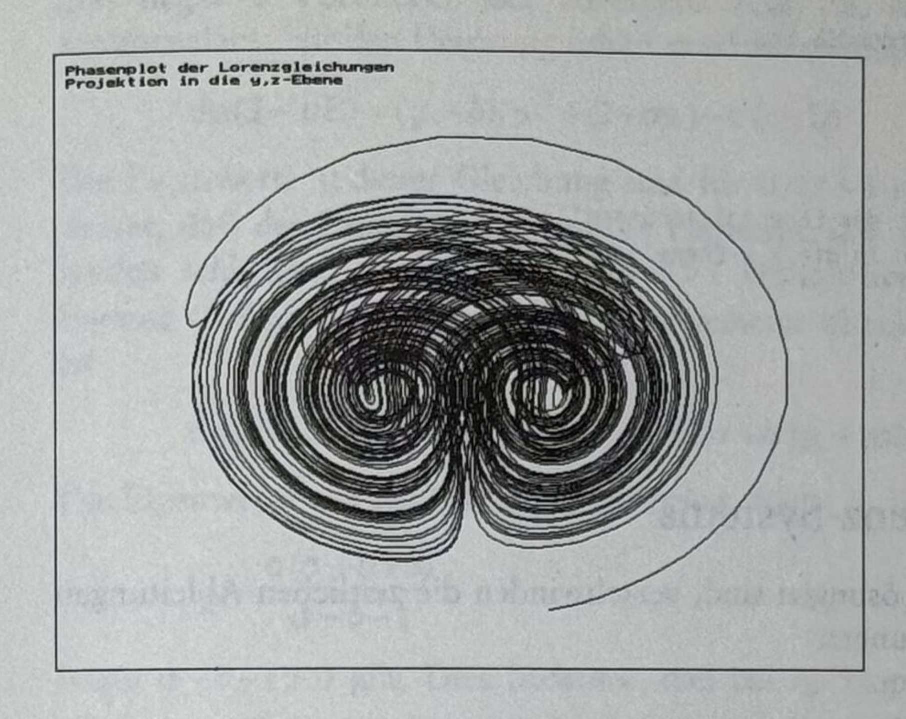


Bild 6.2 Projektion der Lorenzgleichungen in die y-z-Ebene

Ein Programm zum Plotten der Lösungskurven ist das Turbo Pascal-Programm `lorenz.pas`.

```
program lorenz;
uses crt,graph;

const b = 8/3; r=40; sigma=10;
      dt = 0.01;
var dx,dy,dz,x,y,z,t : real;
    Graphdriver,Graphmode:integer;
    first : boolean;

begin
GraphDriver := Detect;
Initgraph(GraphDriver,GraphMode,'\tp\bgi');
SetgraphMode(Graphmode);
rectangle(0,0,639,479);

first := true;
x := 0.01; y := 0.01; z := 0; t := 0;
while t <= 120 do
begin
   dx := sigma*(y-x)*dt;
   dy := (-x*z+r*x-y)*dt;
   dz := (x*y-b*z)*dt;
   x := x + dx;
```

```
    y := y + dy;
    z := z + dz;
    setcolor(1+trunc(t/8));
    if first then
       moveto(round(8*x)+320,470-round(5.5*z))
    else
       lineto(round(8*x)+320,470-round(5.5*z));
    t := t+dt;
    first := false;
    end;
setcolor(15);
moveto(10,10);outtext('Phasenplot der Lorenzgleichungen');
moveto(10,20);outtext('Projektion in die x,z-Ebene');
repeat until keypressed;
closegraph;
textmode(lastmode)
end.
```

6.1 Dynamik des Lorenz-Systems

Da die Fixpunkte stationäre Lösungen sind, verschwinden die zeitlichen Ableitungen und es ergeben sich die Gleichungen

$$0 = -\sigma(x-y)$$
$$0 = (r-z)x-y$$
$$0 = xy-bz$$

Man erkennt sofort, daß $x=y=z=0$ eine Lösung ist, somit ist der Ursprung für alle Parameterwerte ein Fixpunkt. Setzt man $x=y$, so erhält man die weiteren Lösungen

$$x = y = \pm\sqrt{b(r-1)}; \quad z = r-1$$

Für $r>1$ existieren somit die beiden Fixpunkte

$$C^{\pm} = (\pm\sqrt{b(r-1)} \; |\pm\sqrt{b(r-1)}| \; r-1)$$

Die Jacobi-Matrix ergibt sich aus den partiellen Ableitungen

$$\mathbf{J} = \begin{pmatrix} -\sigma & \sigma & 0 \\ r & -1 & 0 \\ y & x & -b \end{pmatrix}$$

Die Spur der Matrix, d.h. die Summe der Diagonalelemente, liefert die Divergenz des zugrundeliegenden Vektorfeldes:

$$div\,\mathbf{F} = -(\sigma+b+1) < 0$$

Da die Divergenz konstant ist, stellt sie die Summe der eindimensionalen Ljapunow-Exponenten dar

$$\lambda_1 + \lambda_2 + \lambda_3 = -(\sigma + b + 1)$$

Das negative Vorzeichen der Divergenz zeigt an, daß sich der Phasenraum zusammenzieht. Für den Ursprung erhält man das charakteristische Polynom

$$\det(\mathbf{J} - \mu\mathbf{E}) = (\mu - b)(\mu^2 + (1+\sigma\mu) - \sigma(r-1))$$

Die Eigenwerte μ dieser Gleichung sind für $0 < r < 1$ alle reell und negativ. Dies bedeutet, daß der Ursprung ein stabiler Fixpunkt, d.h. ein Attraktor ist. Alle Bahnen landen schließlich im Ursprung. Für $r > 1$ verliert der Ursprung seine Stabilität, es kommt zu Gabel-Bifurkationen. Die allgemeine charakteristische Gleichung für $C^{\pm}$ ist

$$\det(\mathbf{J} - \mu\mathbf{E}) = \mu^3 + (1+\sigma + b)\mu^2 + b(r+\sigma)\mu + 2\sigma b(r-1)$$

Die Eigenwerte werden konjugiert komplex für

$$r_H = \frac{\sigma(\sigma + b + 3)}{\sigma - b - 1}$$

wenn $\sigma - b - 1 > 0$ gilt. Dies bedeutet, daß bei r_H Hopf-Bifurkationen einsetzen. Die Fixpunkte $C^{\pm}$ ist somit stabil für $1 < r < r_H$ und instabil für $r > r_H$. Für die Parameterwerte $\sigma = 10$ und $b = \frac{8}{3}$ folgt $r_H = 24.737$. Durch numerische Experimente erhält man folgende Dynamik:

- Für $1 < r < 13.926$ umlaufen alle Trajektorien die Fixpunkte $C^{\pm}$.
- Für $r \approx 13.926$ gibt es eine homokline Bahn; d.h. sie beginnt und endet an einem nichtstabilen Fixpunkt. Dies ist hier der Ursprung.
- Für $13.926 < r < 24.06$ »spiralen« sich alle Bahnen um $C^{\pm}$ und halten sich längere Zeit in der Nähe eines seltsamen Anziehungspunktes auf.
- Für $r > 24.06$ umlaufen noch einige Bahnen $C^{\pm}$, ab $r > 24.737$ winden sich alle Bahnen um den seltsamen Anziehungspunkt, der nun zum Attraktor geworden ist.

Bei $r \approx 22.4$ kündet sich durch Intermittenz nach Manneville und Paumeau bereits das Chaos an; dies zeigt Bild 6.3. Die zugehörigen Trajektorien (Bahnen) können mit dem Turbo C-Programm `trajec.c` erstellt werden.

```
/* trajec.c */
/* Intermittenz bei Lorenzgleichungen */

#include <stdio.h>
#include <graphics.h>
#include <math.h>
#include <conio.h>
```

```
const double sigma=10.,b=2.6667,r=22.4;
const double dt = 0.001;
void main(void)
{
double x,y,z,dx,dy,dz,t;
int gdriver,gmode;
gdriver = 9; gmode = VGAHI; /* VGA-Modus 640 x 480 */
initgraph(&gdriver,&gmode," ");
rectangle(0,0,639,479);
line(10,240,639,240);
setcolor(14);
x = y = 0.8; z = 0.95; t = 0;
do
   {
   dx = sigma*(y-x)*dt;
   dy = (-x*z+r*x-y)*dt;
   dz = (x*y-b*z)*dt;
   x += dx;
   y += dy;
   z += dz;
   putpixel((int)floor(t*6.4),240-(int)floor(10.*y),14);
   t += dt;
   }
while(t<=100.);
moveto(8,8);outtext("Intermittenz bei Lorenz-Trajektoren");
do {} while(!(kbhit()));
closegraph();
return;
}
```

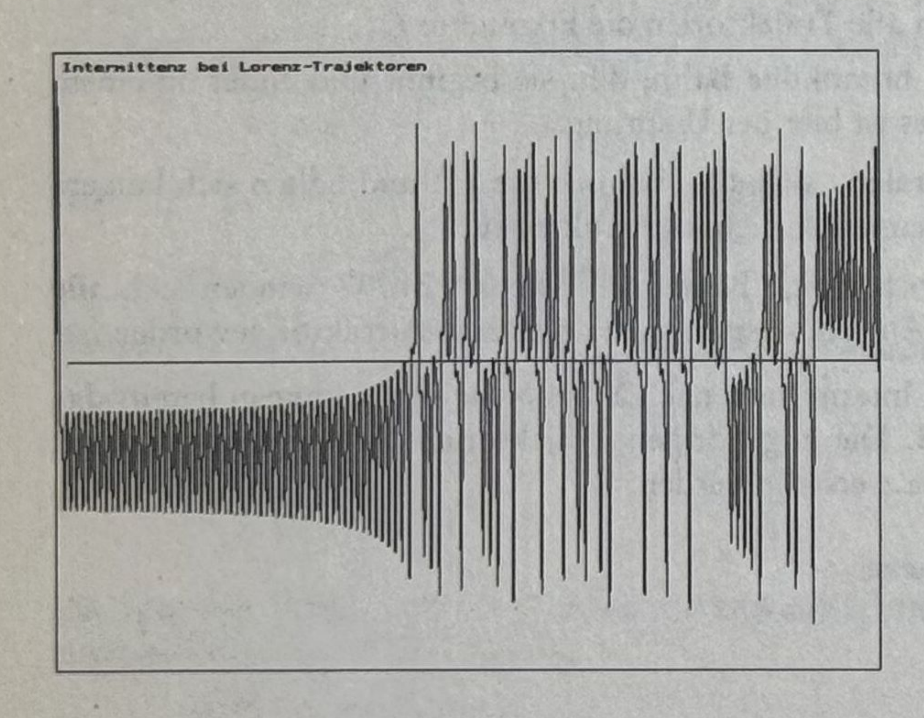


Bild 6.3 Intermittenz bei Lorenz-Gleichungen

6.2 Lorenz-Map

In seiner Arbeit gab Lorenz auch die von ihm gefundene Lorenz-Map an, die in der Literatur auch Return-Map genannt wird. Diese besteht aus dem Plot sukzessiver Maxima der z-Komponente der Lösungskurven. Es ergibt sich hier eine überraschend einfache Abbildung (Bild 6.4), die in ihrem Verhalten der Zeltdach-Funktion ähnlich ist.

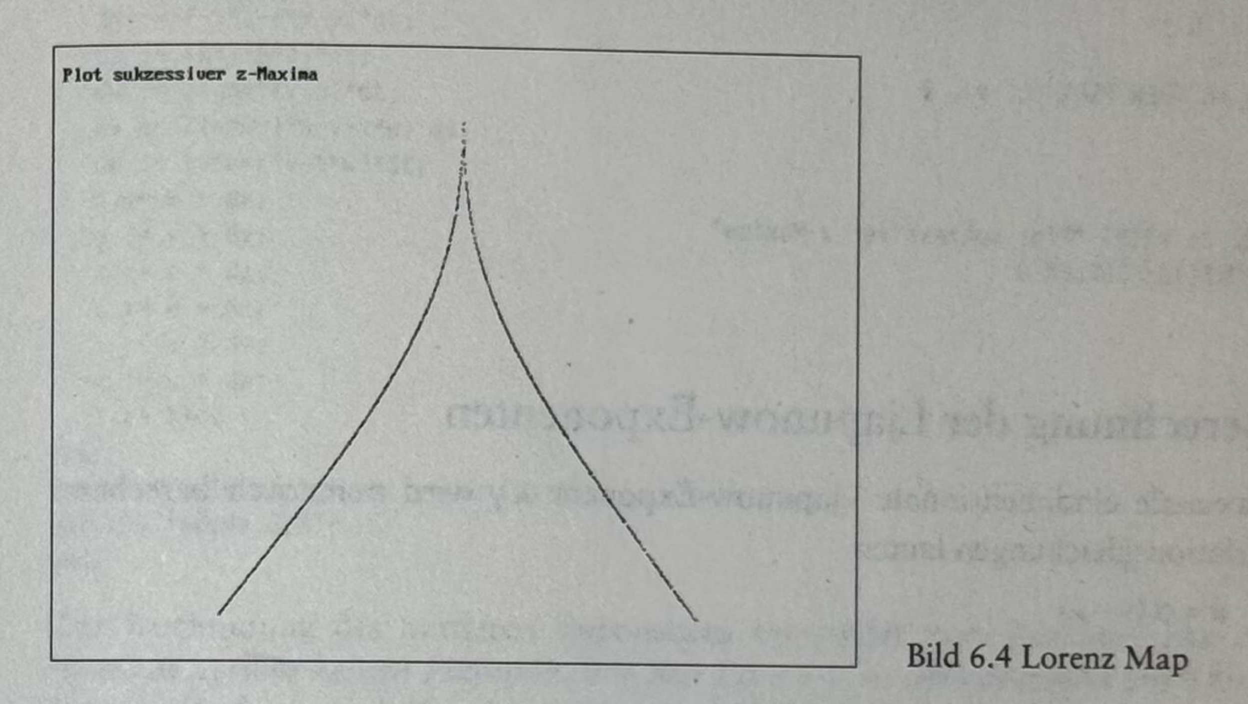


Bild 6.4 Lorenz Map

Es folgt ein Quick Basic-Programm zum Erstellen der Lorenz Map.

```
'lorenz2.bas

CONST sigma = 10: b = 2.66667: r = 28
CONST dt = .01: N = 16000
DIM i AS INTEGER
DIM x, y, z, dx, dy, dz AS DOUBLE
DIM c(N),m(250)

SCREEN 12: CLS
WINDOW (20, 25)-(55, 55)
LINE (20, 25)-(55, 55), , B

x = 1: y = 1: z = 0
FOR k = 1 TO 5
c(0) = z
FOR i = 1 TO N
   dx = sigma * (y - x) * dt
   dy = (r * x - y - x * z) * dt
   dz = (x * y - b * z) * dt
   x = x + dx
   y = y + dy
```

```
  z = z + dz
  c(i) = z
NEXT i
j = 0
FOR i = 1 TO N - 1
IF c(i - 1) < c(i) AND c(i) > c(i + 1) THEN m(j) = c(i): j = j + 1
NEXT i
u = m(2)
FOR i = 3 TO j
  v = m(i)
  IF u < 48 THEN PSET (u, v), 9
  u = v
NEXT i
NEXT k
LOCATE 2, 2: PRINT "Plot sukzessiver z-Maxima"
e$ = INPUT$(1): SCREEN 0
END
```

6.3 Berechnung der Ljapunow-Exponenten

Der maximale eindimensionale Ljapunow-Exponent λ_1 wird numerisch berechnet. Die Variationsgleichungen lauten

$$\dot{u} = \sigma(v-u)$$

$$\dot{v} = (-z+r)u - v - xw$$

$$\dot{w} = yu + xv - bw$$

λ_1 ergibt sich dann aus dem Grenzwert $\lambda_1 = \lim_{n\to\infty} \frac{1}{n} \ln\|(u,v,w)\|$, wobei die Vektornorm prinzipiell frei gewählt werden kann. Das Pascal-Programm `lorlja.pas` liefert für die Parameterwerte $\sigma = 10, b = \frac{8}{3}, r = 28$ den Wert $\lambda_1 = 0.89$. Da die Divergenz gleich

$$div\,\mathbf{F} = -(\sigma + b + 1)$$

ist, folgt hier

$$\lambda_1 + \lambda_2 + \lambda_3 = -(\sigma + b + 1) = -\frac{41}{3}$$

```
program lorlja;
{ Ljapunow-Exponent der Lorenz-Gleichungen }
{$N+}
uses crt,graph;

const dt = 0.001;
      sigma = 10; b=2.6667; r=28;
var i : integer;
    lambda,x,y,z,u,v,w,dx,dy,dz,du,dv,dw,t : extended;
```

```
   Graphdriver,Graphmode:integer;

begin
x := 0.8; y := 0.8; z := 0.8;
u := 0.5; v := 0.5; w := 0.5;
t := 0;
while t<250 do
begin
   dx := sigma*(y-x)*dt;
   dy := (-x*z+r*x-y)*dt;
   dz := (x*y-b*z)*dt;
   du := sigma*(v-u)*dt;
   dv := ((-z+r)*u-v-x*w)*dt;
   dw := (y*u+x*v-b*w)*dt;
   x := x + dx;
   y := y + dy;
   z := z + dz;
   u := u + du;
   v := v + dv;
   w := w + dw;
   t := t+dt
end;
lambda := (ln(abs(u)+abs(v)+abs(w)))/t;
writeln(lambda:6:3);
end.
```

Zur Bestimmung des mittleren Exponenten verwendet man den *Satz*: *Hat ein dissipatives System keinen Fixpunkt,* wie hier für $r > 1$, *so ist mindestens ein Ljapunow-Exponent gleich Null.* Somit gilt $\lambda_2 = 0$. Aus der Divergenz läßt sich daraus noch $\lambda_3 = -14.5$ ermitteln.

Die Attraktoren lassen sich gemäß den Vorzeichen der Ljapunow-Exponenten klassifizieren:

Vorzeichen	**Attraktortyp**	**Dimension**
(–,–,–)	Fixpunkt	0
(0,–,–)	Grenzzyklus	1
(0,0,–)	Torus	2
(+,0,–)	seltsamer Attraktor	fraktal

Damit ist bestätigt, daß für die Wahl $r = 28$ von Lorenz ein seltsamer Attraktor auftritt. Es gibt noch keine allgemein akzeptierte Definition eines seltsamen Attraktors. Die gängigste Definition lautet: *Ein Attraktor heißt seltsam, wenn er sensitiv von den Anfangsbedingungen abhängt.* (vgl. Kap. 19)

Als Beispiel berechnen wir die Kaplan-Yorke-Dimension des Lorenz-Attraktors. Da $\lambda_1 + \lambda_2 > 0$ gilt, folgt $r = 28$. Damit erhalten wir

$$D = j + \frac{\lambda_1}{|\lambda_3|} = 2.06$$

Dies ist auch gleich der Kapazität oder Box-Counting-Dimension des Lorenz-Attraktors. Vergleiche dazu Mandelbrot [17], Seite 210 oder Grassberger in [13]. Für Werte von $r > 250$ verschwindet das chaotische Verhalten des Lorenz-Systems. Dies sieht man am Ljapunow-Diagramm des Lorenz-System (Bild 6.5).

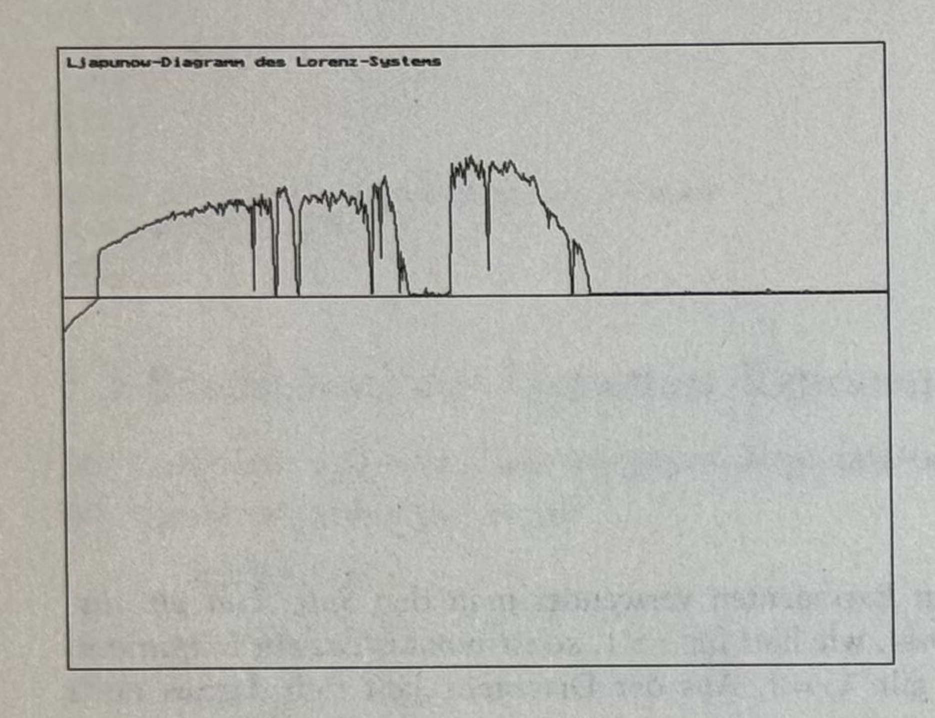
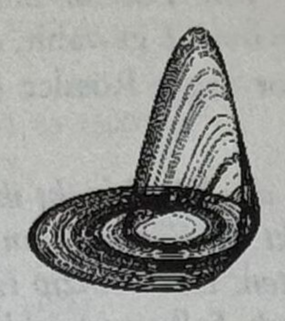


Bild 6.5 Ljapunow-Diagramm für $10 < r < 300$



7 Attraktoren

In diesem Abschnitt werden neben den bereits vorgestellten Attraktoren von Hénon und Lorenz die bekanntesten Systeme mit Attraktoren in Kurzform vorgestellt.

7.1 Rössler-Attraktor

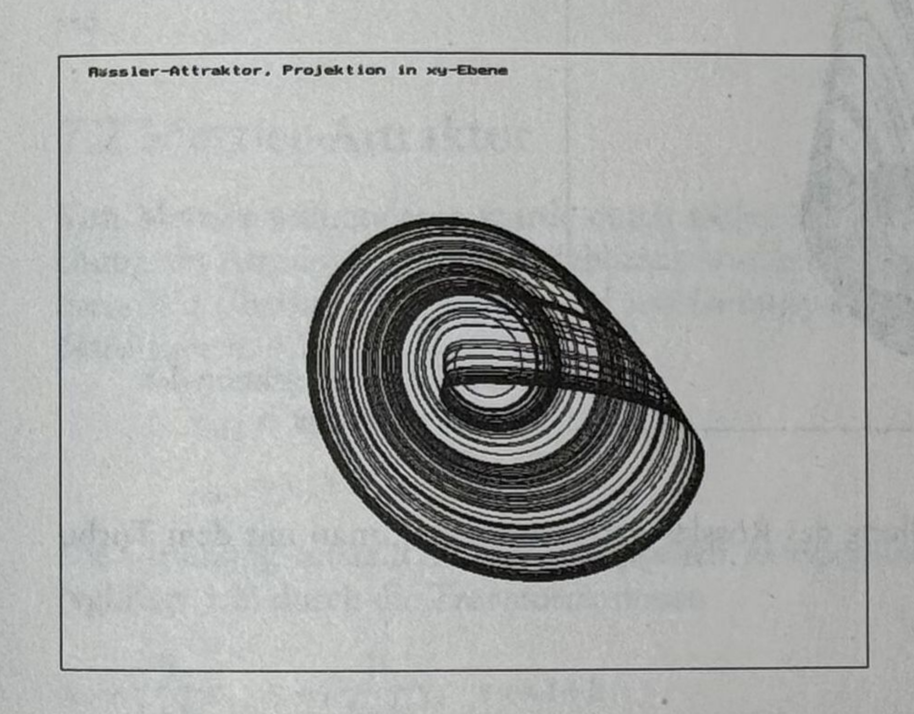


Bild 7.1 Projektion des Rössler-Systems in die xy-Ebene

Durch den Lorenz-Attraktor angeregt, fand der deutschen Mediziner Otto Rössler (*An equation for continuos chaos*, Phys. Lett. 57A,1976) das nach ihm benannte System

$$\dot{x} = -y - z$$
$$\dot{y} = x + ay$$
$$\dot{z} = b + xz - cz$$

dessen Attraktor eine einfachere Dynamik als derjenige von Lorenz aufweist. a, b, c stellen die Parameter des Systems dar; dabei wird oft $a = b = 0.2$ gewählt, c stellt dann den Bifurkationsparameter dar. Über seinen Attraktor sagte Rössler (zitiert nach Gleick):

Er sei wie ein offener Strumpf mit einem Loch am Ende und der Wind bläht ihn auf. Dann sitzt der Wind in der Falle. Gegen ihren Willen bewirkt die Energie nun etwas Produktives, so wie der Teufel in mittelalterlichen Geschichten. Das Prinzip ist, daß die Natur etwas gegen ihren eigenen Willen tut und durch Selbstverwirklichung Schönheit hervorruft.

Wegen der beschriebenen Form heißt das System im Englischen auch *Roessler funnel.*

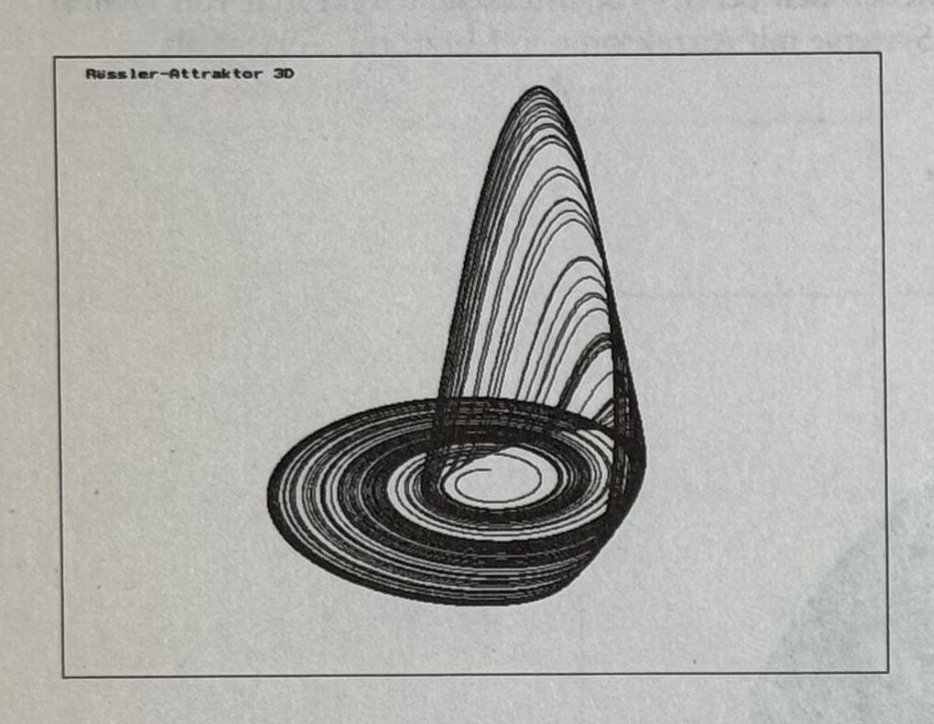


Bild 7.2 3D-Projektion des Rössler-Systems

Eine dreidimensionale Darstellung des Rössler-Attraktors erhält man mit dem Turbo Pascal-Programm `roessler.pas`.

```
program roessler;

uses crt,graph;
const a = 0.25; b = 0.28; c = 5.8;
      dt = 0.01;

var x,y,z,dx,dy,dz : real;
    i : longint;
    Graphdriver,Graphmode:integer;

begin
GraphDriver := Detect;
Initgraph(GraphDriver,GraphMode,'\tp\bgi');
SetgraphMode(Graphmode);
```

```
rectangle(0,0,639,479);
moveto(25,10);outtext('Rössler-Attraktor 3D');

x := -1.0 ; y := 0.2; z := 0;
for i:=1 to 75000 do
   begin
   dx := (-y-z)*dt;
   dy := (x+a*y)*dt;
   dz := (b+z*(x-c))*dt;
   x := x+dx;
   y := y+dy;
   z := z+dz;
   setcolor(1+i div 5400);
   if i=1 then
      moveto(round(12*x+12*y)+330,350-round(-5*x+5*y+9*z))
   else
      lineto(round(12*x+12*y)+330,350-round(-5*x+5*y+9*z));
   end;
repeat until keypressed;
closegraph;
textmode(lastmode)
end.
```

7.2 Metzler-Attraktor

Von Metzler und anderen wurde durch nichtlineare Kopplung der logistischen Gleichung ein Attraktor gefunden. Publiziert wurde die Arbeit in: Metzler W. /Beau W./ Frees W./ Überla A.: *Symmetry and Selfsimilarity with Coupled Logistic Maps*, Z. f. Naturforsch. 42a (1983).

$$x_{n+1} = x_n + h(x_n - x_n^2 + y_n)$$
$$y_{n+1} = y_n + h(y_n - y_n^2 + x_n)$$

Die Gleichung entsteht aus der gekoppelten, zweiparametrigen logistischen Gleichung (vgl.Kap.4.2) durch die Transformationen

$$x \to \frac{h}{1+h}x, \quad y \to \frac{h}{1+h}y, \quad r \to 1+h$$

Numerisch wurde folgende Tabelle der Ljapunow-Exponenten berechnet:

h	λ
0.6780	-0.0053
0.6812	0.1620
0.6840	0.1534
0.6850	0.1702

Damit tritt für $r = 1.648$ Chaos auf. Für diesen Wert bildet sich das Intervall [1.3; 1.3] unter den Metzler-Gleichungen auf sich selbst ab. Der entstehende Attraktor ist dem Eiffelturm ähnlich und heißt deswegen auch *Tour Eiffel de Cassel* nach der Institutsstadt Kassel.

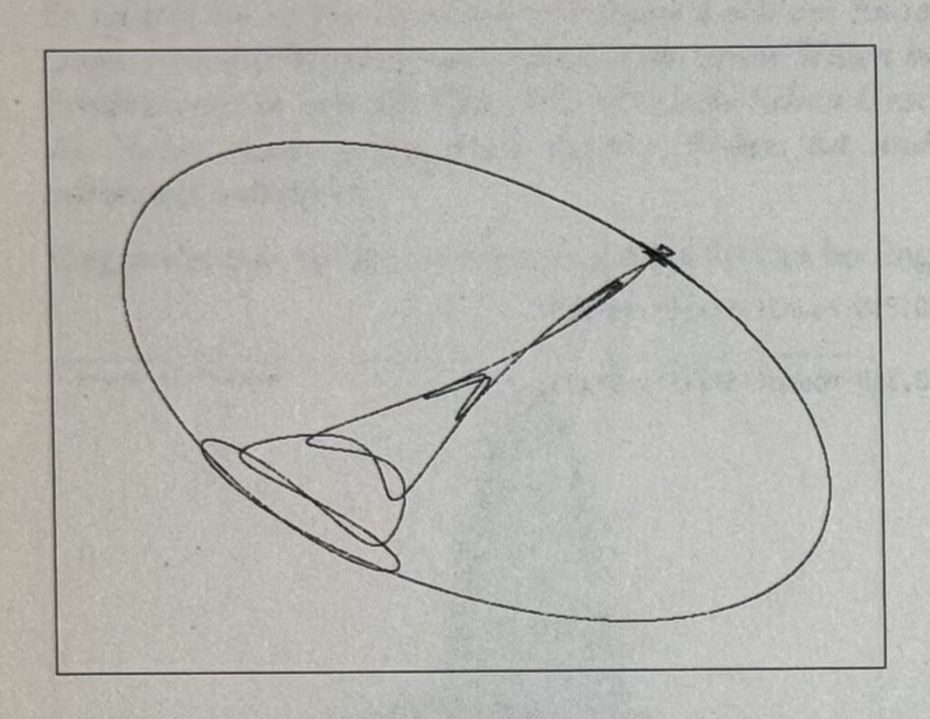


Bild 7.3 Metzler-Attraktor für r=1.684

Die Dynamik des Metzler-Attraktors ist folgende. Für $r = 1.6$ kommt es zu einer Hopf-Bifurkation. Bei $r = 1.65$ wird der Ljapunow-Exponent negativ, das Chaos beginnt. Bei $r = 1.678$ entsteht eine Periode 26. Für $r = 1.678$ findet sich der genannte Attraktor.

Der Metzler-Attraktor kann mit dem Quick Basic-Programm `metzler.bas` erzeugt werden.

```
'metzler.bas
'Metzler-Beau-Attraktor

CONST h = .684
DIM x, y, x1 AS SINGLE
DIM i AS LONG

SCREEN 12: CLS
WINDOW (-1.6, -1)-(3.6, 3)
LINE (-1.6, -1)-(3.6, 3), , B

x = .1: y = .8
FOR i = 1 TO 50000
   x1 = x + h * (x - x * x + y)
   y = y + h * (y - y * y + x)
   x = x1
   IF i > 10 THEN PSET (x, y)
```

```
NEXT i
LOCATE 29, 4: PRINT "Metzler-Attraktor";
e$ = INPUT$(1): SCREEN 0
END
```

7.3 Ikeda-Attraktor

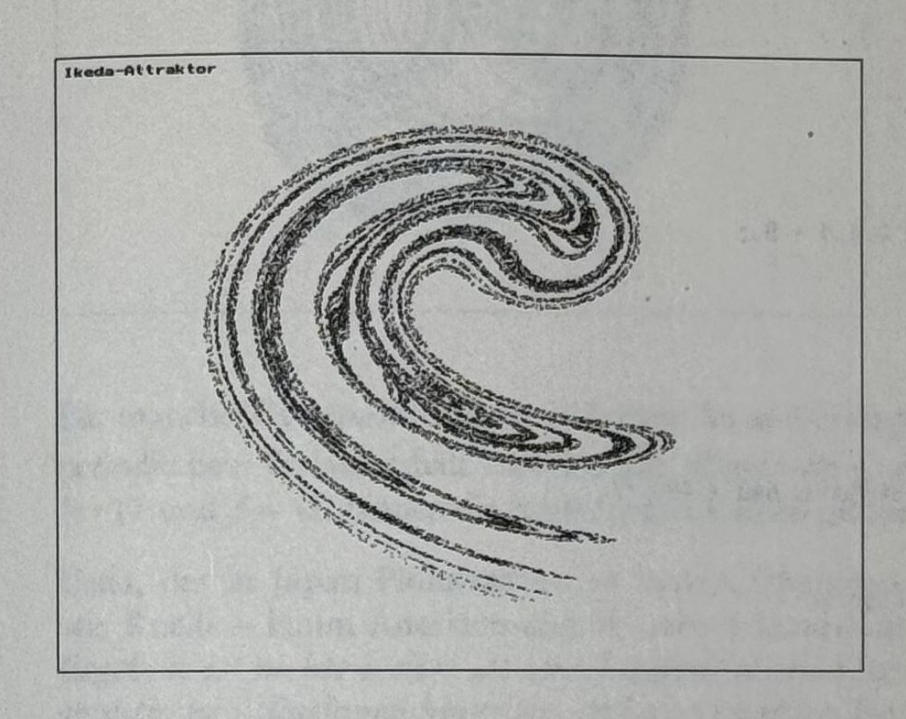


Bild 7.4 Ikeda-Attraktor

Die Ikeda-Gleichung wurde publiziert in: Ikeda I. Akimoto O.: *Instability leading to periodic and chaotic self-pulsations in a bistable optical cave*, Phys. Rev. Lett. 48, (1982). Das Gleichungssystem stammt aus der nichtlinearen Laser-Optik, es hat die komplexe Form

$$z_{n+1} = a - bz_n \exp(i(c - \frac{d}{1+z_n^2}))$$

mit den reellen Parametern a, b, c, d. Die komplexe Exponentialform läßt sich nach der Eulerschen Formel in Real- und Imaginärteil trennen. Dies liefert das zweidimensionale System

$$x_{n+1} = a + b(x_n \cos t - y_n \sin t)$$
$$y_{n+1} = b(x_n \sin t + y_n \cos t)$$

wobei für t gilt

$$t = c - \frac{d}{1 + x_n^2 + y_n^2}$$

Bild 7.4 zeigt ein Phasendiagramm des Ikeda-Attraktors für $a = 0.85, b = 0.9, c = 0.4, d = 9.0$. Das Turbo C-Programm `ikeda.c` stellt den (mut maßlichen) Ikeda-Attraktor am Bildschirm dar.

```
/* ikeda.c */

#include <stdio.h>
#include <math.h>
#include <graphics.h>
#include <conio.h>

const float a = 0.85,b = 0.9,c = 0.4,d = 9.;

void main(void)
{
int i;
double x,x1,y,s,co,t;
int gdriver,gmode;

gdriver = 9; gmode = VGAHI; /* VGA-Modus 640 x 480 */
initgraph(&gdriver,&gmode," ");
rectangle(0,0,639,479);
moveto(8,8);outtext("Ikeda-Attraktor");

x = 0.4; y = 0.5;
for (i=1; i<=25000; i++)
   {
   t = c-d/(1.+ x*x+y*y);
   s = sin(t); co = cos(t);
   x1 = a+b*(x*co-y*s);
   y = b*(x*s+y*co);
   x = x1;
  putpixel(210+floor(180.*x+.5), 210-floor(120.*y+.5), 1+i/6000);
   }
do {} while(!(kbhit()));
closegraph();
return;
}
```

7.4 Ueda-Attraktor

Die Ueda-Gleichungen (Ueda Y./ Akamatsu N.: *Chaotically transitional phenomena in the forced negative resistance oszillator*, IEEE Trans. CS 28 (1981)) stammen aus der Elektronik.

$$\dot{x} = y$$
$$\dot{y} = a(1 - x^2)y - x^3 + b\cos(ft)$$

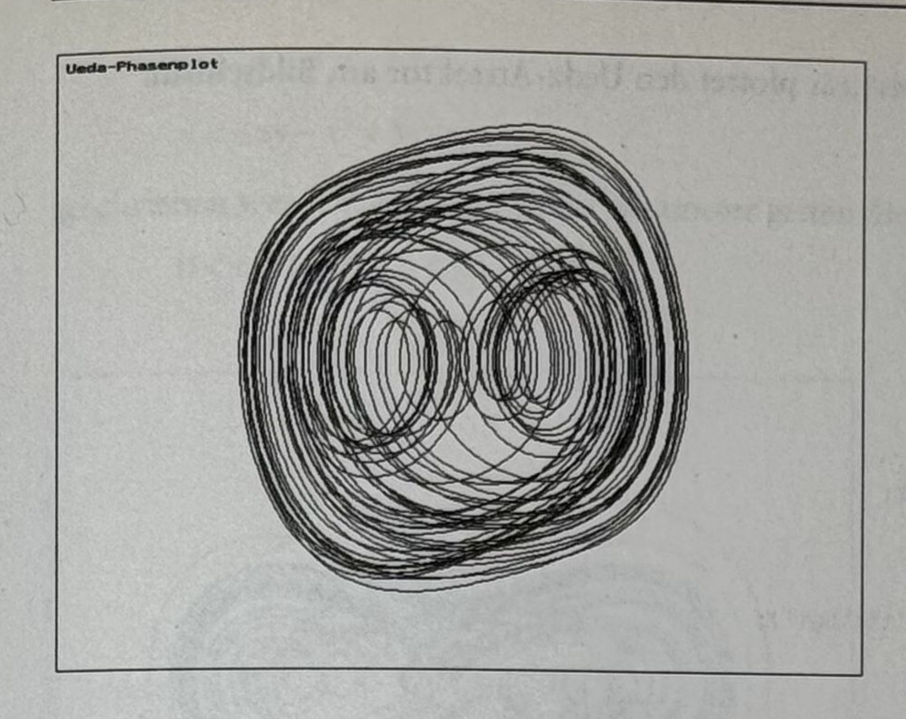


Bild 7.5 Phasenabbildung der Ueda-Gleichungen für b=17 und f=4.0

Für manche Wertepaare von (b, f) kommt keine Schwingungsanregung zustande. Quasiperiodischen Verlauf erhält man für alle Werte von f, wenn b nur klein genug ist. Für $b = 17$ und $f = 4.0$ fanden die beiden Japaner einen (mutmaßlichen) seltsamen Attraktor.

Ueda, der in Japan Pionierarbeit in Sachen Chaosforschung leistete, fand – ähnlich wie Ruelle – kaum Anerkennung bei seinen Landsleuten. Ein Kollege sagte ihm: *Ihr Ergebnis ist nichts weiter als eine fastperiodische Oszillation. Sie dürfen sich keinen egoistischen Illusionen hingeben, daß dies reguläre Zustände sind.*

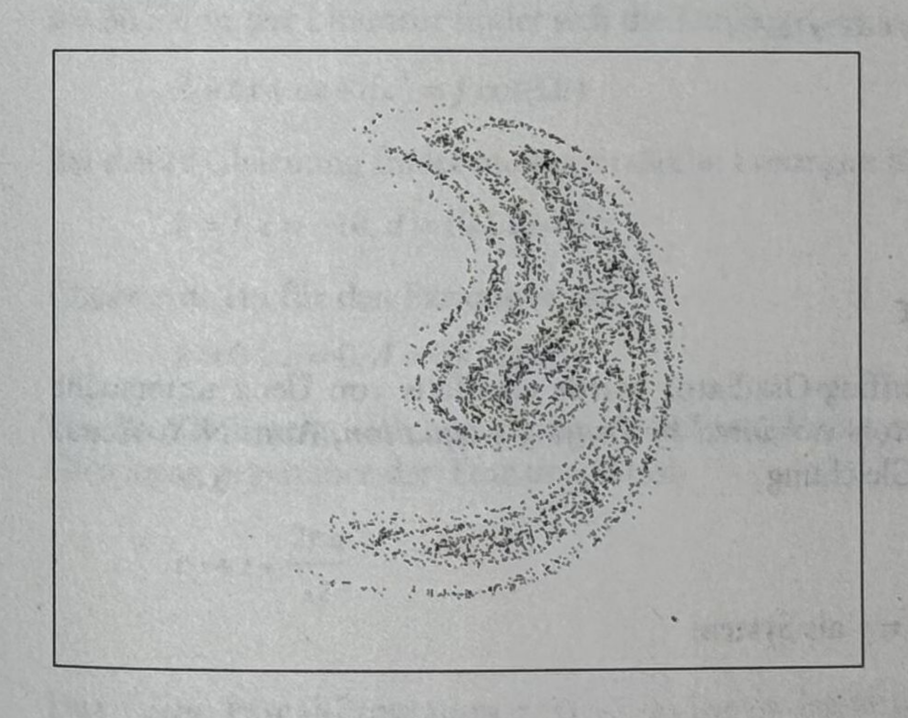


Bild 7.6 Ueda-Attraktor, auch »Japanese attractor« genannt

Das Quick Basic-Programm `ueda2.bas` plottet den Ueda-Attraktor am Bildschirm.

```
program ueda2;
{ Ueda-Attraktor }

{$N+}
uses crt,graph;
const k = 0.1; b=12.0;
      n = 800;
      dt = 2*pi/n;
var x,y,z,dx,dy,dz : double;
    i,j : integer;
    Graphdriver,Graphmode:integer;

begin
GraphDriver := Detect;
Initgraph(GraphDriver,GraphMode,'\tp\bgi');
SetgraphMode(Graphmode);

x:= 2 ; y:= 0; z:=0;
for i:=1 to 5000 do
begin
for j:=1 to n do
   begin
   dx := y*dt;
   dy := (-k*y-x*x*x+b*cos(z))*dt;
   dz := dt;
   x := x+dx;
   y := y+dy;
   z := z+dz;
   end;
   putpixel(round(120*x),230-round(22*y),14);
end;

repeat until keypressed;
textmode(lastmode)
end.
```

7.5 Duffing-Attraktor

Der sinusförmig angeregte Duffing-Oszillator wurde ebenfalls von Ueda untersucht in *Explosion of strange attractors exhibited by Duffing´s equation*, Ann. N.Y. Acad. Sci. 357 (1980). Die Duffing-Gleichung

$$\ddot{x} + a\dot{x} + x^3 = b\cos t$$

kann durch die Substitution $\dot{x} = y$ als System

$$\dot{x} = y$$
$$\dot{y} = -ay - x^3 + b\cos t$$

geschrieben werden. Für die beiden Parameter gelten die Ungleichungen

$$0 < a < 1, \quad 0 < b < 35$$

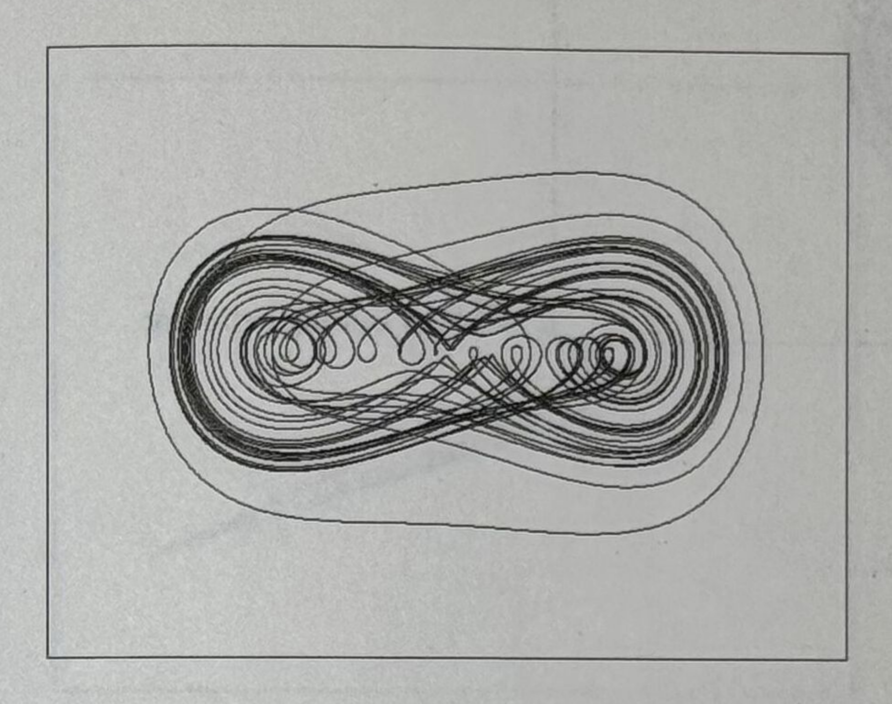


Bild 7.7 Duffing-Oszillator mit $a = 0.2, b = 30.75$

Periodische Lösungen erhält man bei festem $a = 0.2$, z.B., für $b = 16.5$; chaotische für $b = 30.75$. In der Literatur findet sich die Duffing-Gleichung auch in der Form

$$\ddot{x} + b\dot{x} + cx + dx^3 = f\cos(\Omega t)$$

Bei dieser Gleichung findet man periodische Lösungen für

$$b = 1, c = -10, d = 100, \Omega = 3.5$$

Chaos tritt ein für den Parametersatz

$$b = 0.1, c = 0, d = 100, \Omega = 1$$

Ein Attraktor zeigt sich beim Poincaré-Schnitt, der sich ergibt durch die Invarianz der Gleichung gegenüber der Transformation

$$t \to t + \frac{2\pi n}{\Omega}$$

Das Turbo Pascal-Programm `duffing3.pas` liefert den Poincaré-Schnitt des Attraktors.

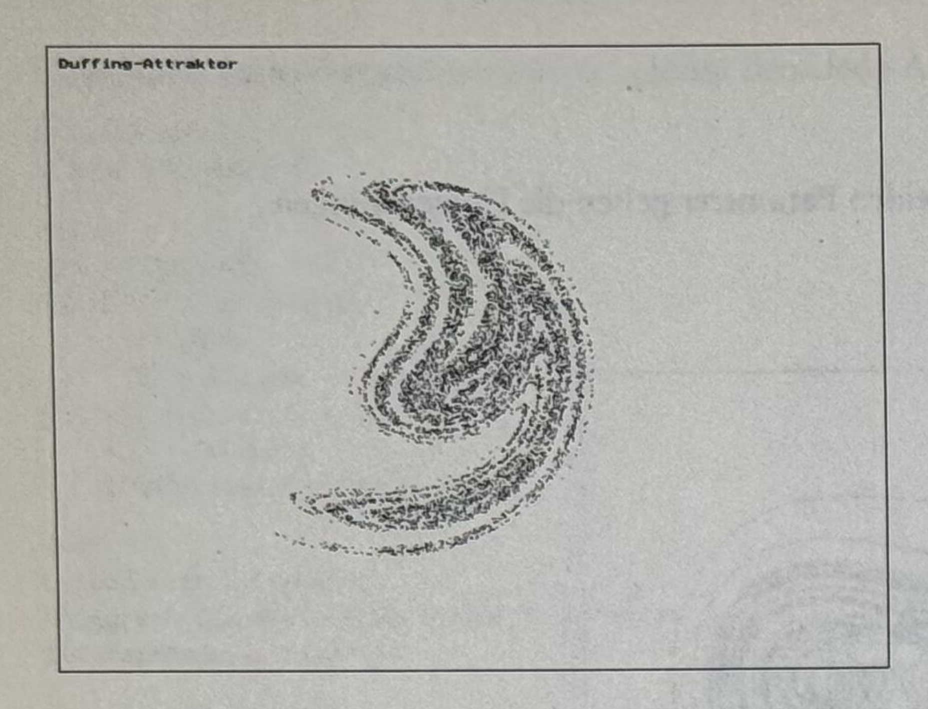


Bild 7.8 Duffing-Attraktor

```
program duffing3;
{$N+,E+}

uses crt,graph;
const b=0.1; c=0; f=12; omega = 1;
      n = 800; p2 = 2*pi;
      dt = 2*pi/(omega*n);
var x,y,z,dx,dy,dz : double;
    i,j : longint;
    Graphdriver,Graphmode:integer;

begin
GraphDriver := Detect;
Initgraph(GraphDriver,GraphMode,'\tp\bgi');
SetgraphMode(Graphmode);
rectangle(0,0,639,479);
moveto(10,10);outtext('Duffing-Attraktor');
x:=-1.0 ; y:=1.0; z:=0; { Startwerte }
for i:=0 to 8000 do

begin
   for j:=1 to n do
   begin
   dx := y*dt;
   dy := (-(x*x*x+c*x+b*y)+f*cos(p2*z))*dt;
   dz := omega/p2*dt;
   x := x+dx;
   y := y+dy;
   z := z+dz;
   end;
   if i>50 then putpixel(round(60*x)+140,240-round(17*y),15)
end;
```

```
repeat until keypressed;
closegraph;
textmode(lastmode)
end.
```

7.6 Lozi-Gleichung

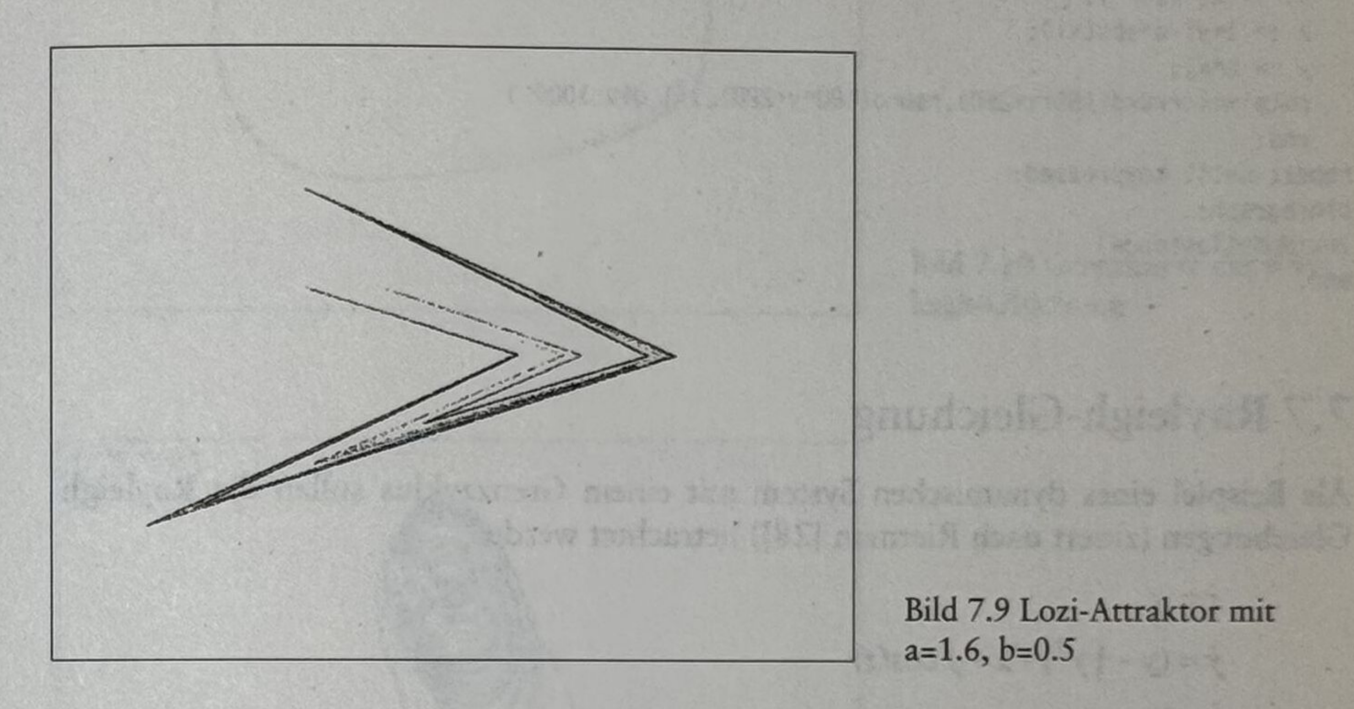


Bild 7.9 Lozi-Attraktor mit a=1.6, b=0.5

Die Lozi-Gleichung, publiziert in *Un attracteur étrange(?) du type de Hénon*, J. de Physique, 39 (1978), kann als Linearisierung der Hénon-Abbildung aufgefaßt werden

$$x_{n+1} = 1 + y_n - a|x_n|$$
$$y_{n+1} = bx_n$$

mit $a, b > 0$. Numerische Berechnungen sprechen für die Existenz eines seltsamen Attraktor für die Parameterwerte $a = 1.7, b = 0.5$. Der (mutmaßliche) Lozi-Attraktor kann mit dem Turbo Pascal-Programm `lozi.pas` geplottet werden.

```
program lozi;
{$N+}

uses crt,graph;
const a=1.7; b=0.5;
var i:longint;
    x,x1,y,y1,u:double;
    Graphdriver,Graphmode:integer;

begin
GraphDriver := Detect;
```

```
Initgraph(GraphDriver,GraphMode,' ');
SetgraphMode(Graphmode);
cleardevice;
setbkcolor(15);

x := 0.5; y:= 0.5;
for i:=1 to 10000 do
   begin
   x1 := x; y1:= y;
   x := 1+y1-a*abs(x1);
   y := b*x1;
   putpixel(round(180*x+280),round(180*y+220),1+i div 1000 )
   end;
repeat until keypressed;
closegraph;
TextMode(lastmode)
end.
```

7.7 Rayleigh-Gleichung

Als Beispiel eines dynamischen System mit einem Grenzzyklus sollen die Rayleigh-Gleichungen (zitiert nach Rietman [28]) betrachtet werden

$$\dot{x} = y$$
$$\dot{y} = (y - \tfrac{1}{3}y^3) - x + f\cos(z)$$
$$\dot{z} = \omega$$

Für die Parameter $\omega = 1, f = 1$ erhält man die Grenzkurve von Bild 7.10. Das Programm soll als Übung geschrieben werden.

7.8 Tomita-Gleichung

Von Tomita und Kai wurde die Brüsselator-Gleichung (vgl. Kapitel 17.2) um einen sinusfömigen Term ergänzt. Die Gleichung wurde publiziert in Tomita K./ Kai T.: *Stroboscopic phase portrait and strange attractors*, Phys. Lett. 66A (1987)

$$\dot{x} = a + x^2 y - bx - x + c\cos(ft)$$
$$\dot{y} = bx - x^2 y$$

Standardmäßig werden die Werte $a = 0.4, b = 1.2, c = 0.05$ gewählt. Für wachsende Werte von f erhält man Perioden-Verdopplungen, beginnend mit $f = 0.8$. Für $f = 0.95$ wird schließlich Chaos erreicht.

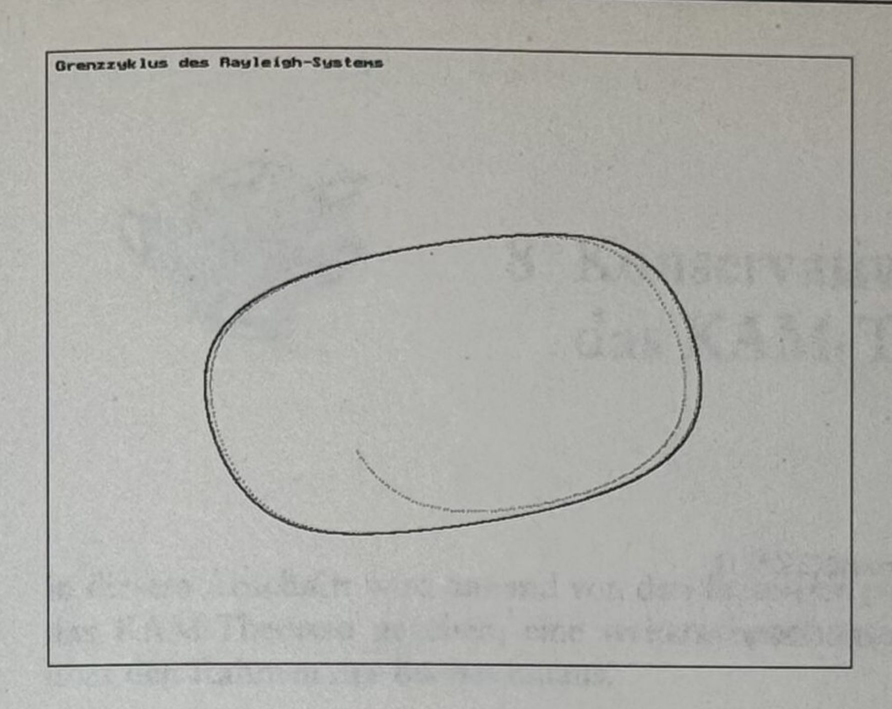


Bild 7.10 Grenzkurve der Rayleigh-Gleichung

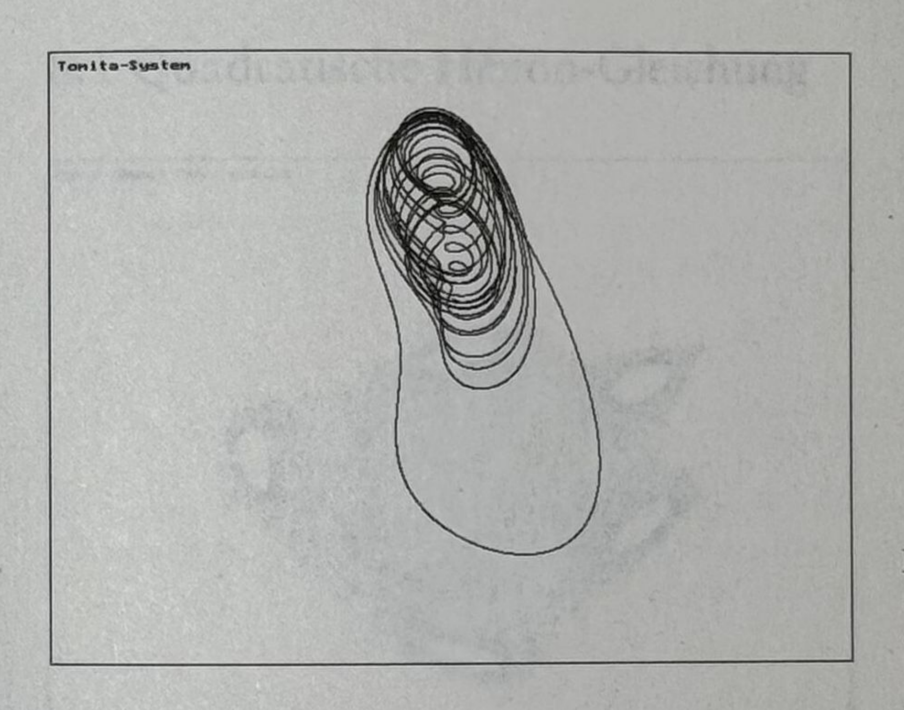


Bild 7.11 Tomita-System mit $a = 0.4,\ b = 1.2,\ c = 0.05,$ $f = 0.95$

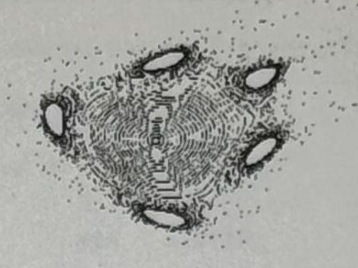
```
program tomita;
uses crt,graph;

const a = 0.4; b = 1.2; c=0.05; f=0.95;
      dt = 0.01;
var x,y,z,dx,dy,dz : real;
    i : longint;
     Graphdriver,Graphmode:integer;

begin
GraphDriver := Detect;
Initgraph(GraphDriver,GraphMode,'\tp\bgi');
```

```
SetgraphMode(Graphmode);
rectangle(0,0,639,479);

x:= 0.5 ; y := 3.0; z:=0;
for i:=0 to 25000 do
   begin
   dx := (a+x*x*y-b*x-x+c*cos(z))*dt;
   dy := (b*x-x*x*y)*dt;
   dz := dt;
   x := x+dx;
   y := y+dy;
   z := z+dz;
   setcolor(1+i div 1000);
   if i=0 then
   moveto(round(400*x)+150,1100-round(330*y))
   else
   lineto(round(400*x)+150,1100-round(330*y))
   end;
 repeat until keypressed;
 closegraph;
 textmode(lastmode)
 end.
```



8 Konservative Systeme und das KAM-Theorem

In diesem Abschnitt wird anhand von drei Beispielen eine anschauliche Einführung in das KAM-Theorem gegeben; eine weitergehende mathematische Darstellung führt über den Rahmen des Buches hinaus.

8.1 Quadratische Hénon-Gleichung

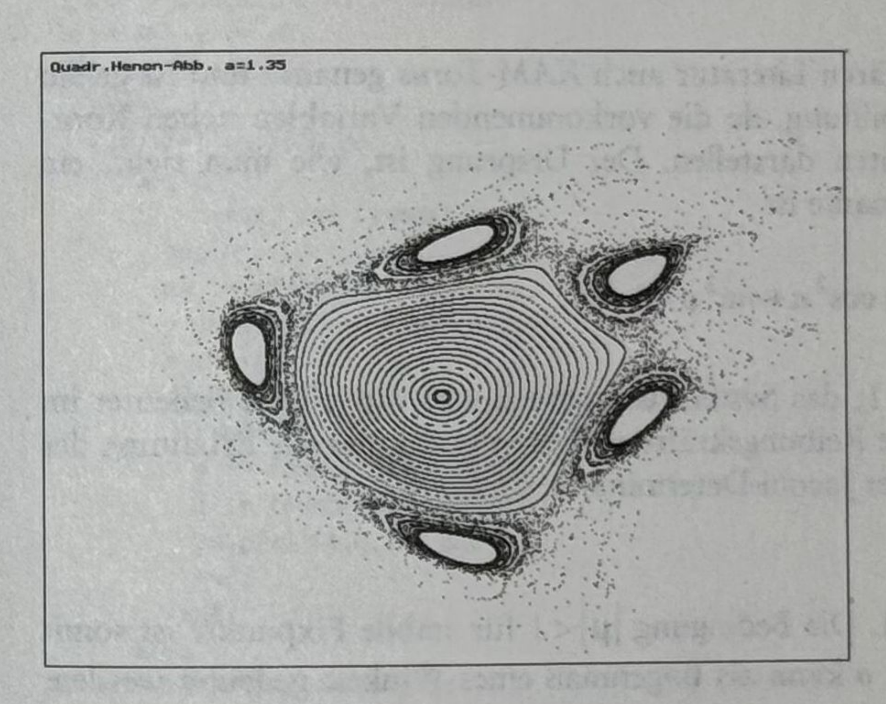


Bild 8.1 Quadratische Hénon-Gleichung für $a = 1.35$

Die quadratische Hénon-Gleichung lautet

$$x_{n+1} = x_n \cos a - (y_n - x_n^2) \sin a$$

$$y_{n+1} = x_n \sin a + (y_n - x_n^2) \cos a$$

publiziert in Henon M.: *Numerical Study of a quadratic area-preserving mappings*, Q. Appl. Math. 50, 1976.

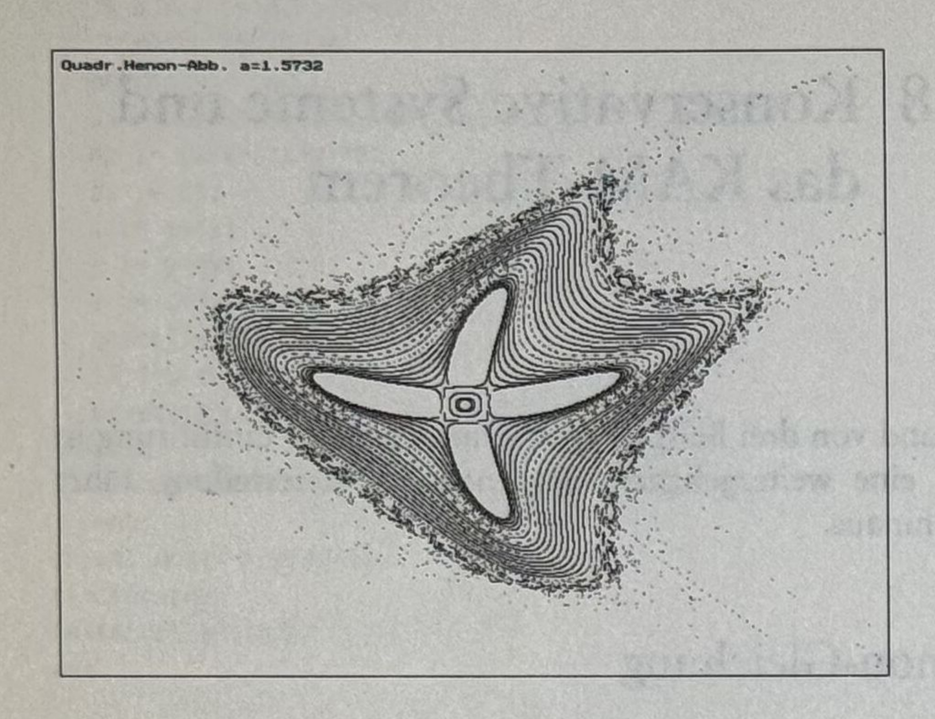


Bild 8.2 Quadratische Hénon-Gleichung für $a = 1.5237$ mit 1:4 Resonanz

Das System wird in der populären Literatur auch *KAM-Torus* genannt und ist genau genommen eine Poincaré-Abbildung, da die vorkommenden Variablen neben Koordinaten auch Geschwindigkeiten darstellen. Der Ursprung ist, wie man sieht, ein Fixpunkt. Die Jacobi-Determinante ist

$$\det \mathbf{J} = \begin{vmatrix} \cos a & -\sin a \\ \sin a & \cos a \end{vmatrix} = \cos^2 a + \sin^2 a = 1$$

im Ursprung konstant gleich 1; das System ist somit konservativ. Dies bedeutet im physikalischen Sinn, daß keine Reibungskräfte auftreten und somit die Erhaltung der Energie gilt. Die Eigenwerte der Jacobi-Determinante sind

$$\mu = e^{\pm ia}$$

und haben daher den Betrag 1. Die Bedingung $|\mu| < 1$ für stabile Fixpunkte ist somit nicht erfüllbar. Der Parameter a kann als Bogenmaß eines Winkels gedeutet werden; interessante Kurven erhält man durch Werte von a in der Nähe von $\frac{\pi}{2}$.

Bilder 8.1 und Farbtafel 10 zeigen die quadratische Hénon-Gleichung für $a = 1.35$, Bild 8.2 für $a = 1.5732$. Wie im Abschnitt 8.4 gezeigt wird, kann der Parameter a als Maß für eine Störung des Systems gedeutet werden. Für $a = 1.35$ ist also die Störung noch klein und betrifft nur die Randgebiete. Dagegen sind die Kreise für $a = 1.5732$ zu einer 1:4-Resonanz aufgebrochen.

Das Turbo Pascal-Programm `henon.pas` erzeugt die Graphik für die quadratische Hénon-Abbildung.

```
program henon; { quadratische Henon-Abbildung }

uses crt,graph;

const a = 1.35;
      h = 0.02;
      max = 1.0e10;

var i,j,p,q : integer;
    x0,x,x1,y0,y,y1,cosa,sina : real;
    Graphdriver,Graphmode:integer;

begin
GraphDriver := Detect;
Initgraph(GraphDriver,GraphMode,'\tp\bgi');
SetgraphMode(Graphmode);
cleardevice;
rectangle(0,0,639,479);
moveto(8,8);outtext('Quadr.Henon-Abb. a=1.35');

x0 := 0.01; y0 := -0.02;
cosa := cos(a); sina := sin(a);
x := x0; y := y0;
for j:=1 to 50 do
  begin
  for i:=1 to 1500 do
     begin
     if (x<max) and (y<max) then
       begin
       x1 := x*cosa-(y-x*x)*sina;
       y := x*sina+(y-x*x)*cosa;
       x := x1;
       if (abs(x+1.3)<3) and (abs(1.5-y)<3) then
         begin
         p := round((x+1.3)*240);
         q := round((1.5-y)*180);
         putpixel(p,q,1+j mod 16);
         end;
       end;
     end;
  x := x0+j*h; y := y0+j*h;
  end;
repeat until keypressed;
closegraph;
TextMode(lastmode);
end.
```

8.2 Das KAM-Theorem

Das KAM-Theorem, benannt nach den russisch-deutschen Mathematikern Kolmogorow (1954), Arnold (1963) und Moser (1973), löst ein Problem, das von Poincaré aufgeworfen wurde.

Die Bewegung eines konservativen Systems von n Freiheitsgraden kann in der klassischen Mechanik beschrieben werden durch eine Hamilton-Funktion H

$$H = H(p_1, p_2, \ldots, p_n, q_1, q_2, \ldots, q_n)$$

dabei stellen die Variablen p_j die Ortskoordinaten und q_j die Impulse dar. In einigen Fällen, die man in jedem Lehrbuch der theoretischen Mechanik findet, können die Hamiltonschen Bewegungsgleichungen

$$\dot{q}_j = \frac{\partial H}{\partial p_j}; \quad \dot{p}_j = \frac{\partial H}{\partial q_i}$$

exakt integriert werden. Dies ist der Fall, wenn eine Variablen-Transformation

$$(p_1, p_2, \ldots, p_n, q_1, q_2, \ldots, q_n) \rightarrow (r_1, r_2, \ldots, r_n, s_1, s_2, \ldots, s_n)$$

so ausgeführt werden kann, daß gilt

$$\frac{\partial H}{\partial r_j} = \omega_j(r_1, r_2, \ldots, r_n); \quad \frac{\partial H}{\partial s_j} = 0$$

Die Variablen s_j heißen dann zyklisch. In den allermeisten Fällen ist dies jedoch nicht möglich.

Die von Poincaré aufgeworfene Fragestellung war: Angenommen, H_0 ist eine integrierbare Hamilton-Funktion. Was läßt sich dann für die Lösungen eines gestörten Systems mit der Hamiltonfunktion

$$H = H_0 + \varepsilon H_1(p_j, q_j)$$

aussagen, wenn der Parameterwert ε als klein und somit εH_1 als kleine Störung angesehen werden kann?

Das Verhalten des ungestörten Systems ist bestimmt durch die Gleichungen

$$\dot{r}_j = \frac{\partial H}{\partial s_k} = 0 \Rightarrow r_j = const$$

$$\dot{s}_j = \frac{\partial H}{\partial r_k} = \omega_j \Rightarrow s_j = \omega_j t + s_{j0}$$

Die Größen ω_j werden in der Regel als Frequenzen interpretiert, die Bewegung verläuft hier auf den Phasenraum-Flächen $r_j = const$, die als invariante Tori bezeichnet werden. Das KAM-Theorem besagt nun anschaulich, daß das Hamilton-System bei kleinen Störungen stabil bleibt bis auf schmale instabile Bänder, die einer Resonanzfrequenz des ungestörten und des gestörten Systems entsprechen. Ein Torus mit der ungestörten Frequenz ω_0 und der gestörten Frequenz $\omega_0(\varepsilon)$ überlebt bei größerwerdenden Störungen umso länger, je weniger das Frequenzverhältnis $\omega_0 : \omega_0(\varepsilon)$ im rationalen Verhältnis steht.

8.3 Hénon-Heiles-System

Als Beispiel eines Hamilton-Systems werden die Gleichungen von Hénon-Heiles besprochen, die 1964 in *Astron. J.* 69, 73 publiziert wurden. Die Hamilton-Funktion des Systems zweier galaktischer Cluster lautet

$$H(p_1, p_2, q_1, q_2) = \tfrac{1}{2}\left(p_1^2 + p_2^2 + q_1^2 + q_2^2\right) + q_1^2 q_2 - \tfrac{1}{3} q_2^3$$

Die Hamilton-Funktion dieses nicht-integrablen System ist nicht skaleninvariant, daher stellt die Energie E einen Bifurkations-Parameter dar. Es zeigt sich, daß sich für $E = \frac{1}{12}$ geschlossene Bahnen ergeben. Für $E = \frac{1}{8}$ erhält man den Grenzfall, daß bereits einige Bahnen irregulär sind. Im Fall $E = \frac{1}{6}$ ist das Chaos fast vollständig.

Gibt man die Startwerte $(p_{10}, p_{20}, q_{10}, q_{20})$ vor, so ist damit die (Gesamt)-Energie

$$E = H(p_{10}, p_{20}, q_{10}, q_{20})$$

bestimmt. Nach dem Energiesatz ist E konstant und somit ein Integral der Bewegung. Zur Darstellung wählt man eine geeignete Poincaré-Abbildung. Die Bahnen liegen alle auf der 3-dimensionalen Energie-Hyperfläche $E = H(\mathbf{p}, \mathbf{q})$ als Unterraum des vierdimensionalen Phasenraums. Als Poincaré-Abbildung betrachten wir die Durchstoßpunkte der Bahnen mit der (p_2, q_2)-Ebene für $q_1 = 0 \wedge p_2 > 0$.

Diese Bedingung führt nun dazu, daß alle Durchstoßpunkte innerhalb einer vorgegebenen Randkurve liegen, die vom Energiewert E abhängt. Diese Randkurven erhält man aus den Hamilton-Gleichungen hier für $p_1 = q_1 = 0$.

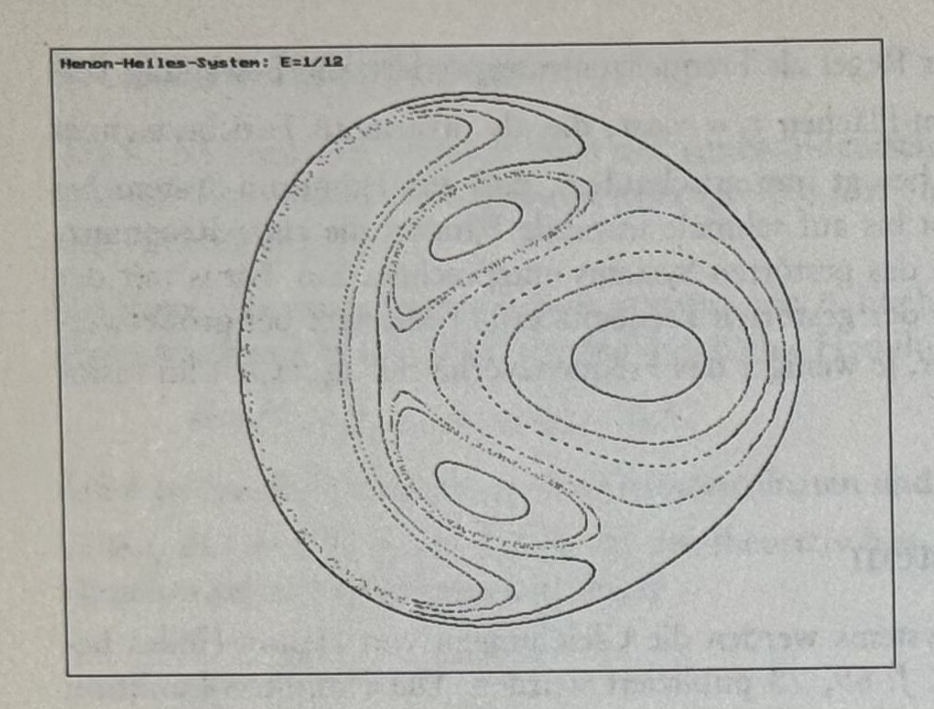


Bild 8.3 Poincaré-Abbildung des Hénon-Heiles-Systems für $E = \frac{1}{12}$

Die Hamilton-Gleichungen

$$\dot{p}_j = \frac{\partial H}{\partial q_j}; \quad \dot{q}_j = \frac{\partial H}{\partial p_j}$$

liefern hier die Bewegungsgleichungen

$$\dot{p}_1 = -q_1 - 2q_1q_2$$
$$\dot{p}_2 = -q_2 - q_1^2 + q_2^2$$
$$\dot{q}_1 = p_1$$
$$\dot{q}_2 = p_2$$

Das folgende Turbo Pascal-Programm `heiles.pas` erzeugt die Poincaré-Abbildung für den interessanteren Fall $E = \frac{1}{12}$. Um verschiedene Bahnkurven zu erzeugen, wurden im Programm 10 verschiedene Anfangswerte $(p_{10}, p_{20}, q_{10}, q_{20})$ vorgegeben. Zusätzlich wird die Randkurve mittels einer separaten Prozedur geplottet (Bild 8.3).

```
program heiles;
{$N+}
uses crt, graph;
const t = 0.01;
var x1,x2,x3,x4 : array[1..12] of double;
    p1,p2,q1,q2,t2,t3,count : double;
    graphdriver,graphmode,j : integer;

procedure randkurve;
var x,y:double;
begin
```

```
y := -0.1;
while y <= 0.5 do
  begin
  x := 1/6-y*y*(1+2*y/3);
  if x>0 then
    begin
    putpixel(320-trunc(500*y),240+trunc(500*sqrt(x)),14);
    putpixel(320-trunc(500*y),240-trunc(500*sqrt(x)),14);
    end;
  y := y+0.001;
  end;
y := -0.5;
while y <= 0.1 do
  begin
  x := 1/6-y*y*(1+2*y/3);
  if x>0 then
    begin
    putpixel(320-trunc(500*y),240+trunc(500*sqrt(x)),14);
    putpixel(320-trunc(500*y),240-trunc(500*sqrt(x)),14);
    end;
  y := y+0.001;
  end;
end;

begin
graphdriver := detect;
Initgraph(graphdriver,graphmode,'');
cleardevice;
rectangle(0,0,639,479);
randkurve;
moveto(8,8);outtext('Henon-Heiles-System: E=1/12');
t2 := t*t; t3 := t*t2;
x1[2] := 0; x2[2] := sqrt(1/8); x3[2] := sqrt(1/24); x4[2] := 0;
x1[3] := sqrt(1/18); x2[3] := sqrt(1/18); x3[3] := sqrt(1/18); x4[3] := 0;
x1[4] := sqrt(1/12); x2[4] := sqrt(1/12); x3[4] := 0; x4[4] := 0;
x1[5] := 0; x2[5] := 0; x3[5] := sqrt(1/6); x4[5] := 0;
x1[6] := 0; x2[6] := 0; x3[6] := sqrt(13/270); x4[6] := 1/3;
x1[7] := 0; x2[7] := 0; x3[7] := sqrt(11/144); x4[7] := 0.25;
x1[8] := sqrt(1/36); x2[8] := 1/3; x3[8] := sqrt(1/36); x4[8] := 0;
x1[9] := sqrt(1/6); x2[9] := 0; x3[9] := 0; x4[9] := 0;
x1[10] := sqrt(1/15); x2[10] := 0; x3[10] := sqrt(1/10); x4[10] := 0;
x1[11] := 0.1; x2[11] := 0; x3[11] := sqrt(47/300); x4[11] := 0;
x1[12] := sqrt(11/75); x2[12] := 0.1; x3[12] := 0.1; x4[12] := 0;
count := 0;
for j:= 1 to 11 do
begin
while count <= 2000 do
begin
p1 := x1[j+1]; q1 := x2[j+1]; p2 := x3[j+1]; q2 := x4[j+1];
x1[j+1] := (t3*(2*p2*p2*p2+2*p2*q2*q2+6*p2*q2+p2-4*p1*q1)+
      t2*(-6*p2*q1-6*q2*p1-3*p1)+t*(-12*p2*q2-6*p2)+6*p1)/6;
```

```
x2[j+1] := (t3*(2*p2*p2*q2+3*p2*p2+2*q2*q2*q2-3*q2*q2+q2-2*p1*p1+2*q1*q1)+
      t2*(-6*p2*p1+6*q2*q1-3*q1)+t*(-6*p2*p2+6*q2*q2-6*q2)+6*q1)/6;
x3[j+1] := (t3*(-2*p2*q1-2*q2*p1-p1)+t2*(-6*p2*q2-3*p2)+t*6*p1+6*p2)/6;
x4[j+1] := (t3*(-2*p2*p1+2*q2*q1-q1)+t2*(-3*p2*p2+3*q2*q2-3*q2)+t*6*q1+6*q2)/6;
  if (p2*x3[j+1]<0) and (p2>0) then
     begin
     putpixel(round(500*x4[j+1])+320,240-round(500*x2[j+1]) ,j);
     putpixel(round(500*x4[j+1])+320,240+round(500*x2[j+1]),j);
     end;
count := count+t;
end;
count := 0;
end;
repeat until keypressed;
closegraph
end.
```

8.4 Twist-Map

Die von J. Moser eingeführte Drehabbildung, *Twist-Map* genannt, ist definiert durch die Abbildungen

$$x_{n+1} = x_n \cos a - y_n \sin a$$
$$y_{n+1} = x_n \sin a + y_n \cos a$$

Diese Abbildung hat dieselbe Jacobi-Determinante wie die quadratische Hénon-Gleichung im Ursprung. Die Twist-Map kann leicht in Polarkoordinaten (θ, r) überführt werden. Die Polarform lautet

$$\theta_{n+1} = \theta_n + a \mod 2\pi$$
$$r_{n+1} = r_n$$

Die Gleichungen zeigen, daß die radialen Abstände gleich bleiben und das Winkelargument bei jeder Drehung um a wächst. Die invarianten Kurven sind daher Kreise bzw. Tori. Ist a eine rationale Zahl, so wird irgendwann der Startpunkt der Bewegung wieder erreicht. Diese Bewegungen beschreiben die sog. rationalen Tori. Entsprechend findet die Bewegung bei irrationalem a auf den irrationalen Tori statt.

Das KAM-Theorem besagt nun, daß bei einer kleinen Störung nur die irrationalen Tori erhalten bleiben, die *hinreichend* irrational sind. Hinreichend irrational heißt eine Zahl a hier, wenn die Ungleichung

$$\left| a - \frac{m}{n} \right| \geq \frac{c}{n^{\mu}}$$

für alle natürlichen Zahlen m,n besteht mit $\mu > 2$.

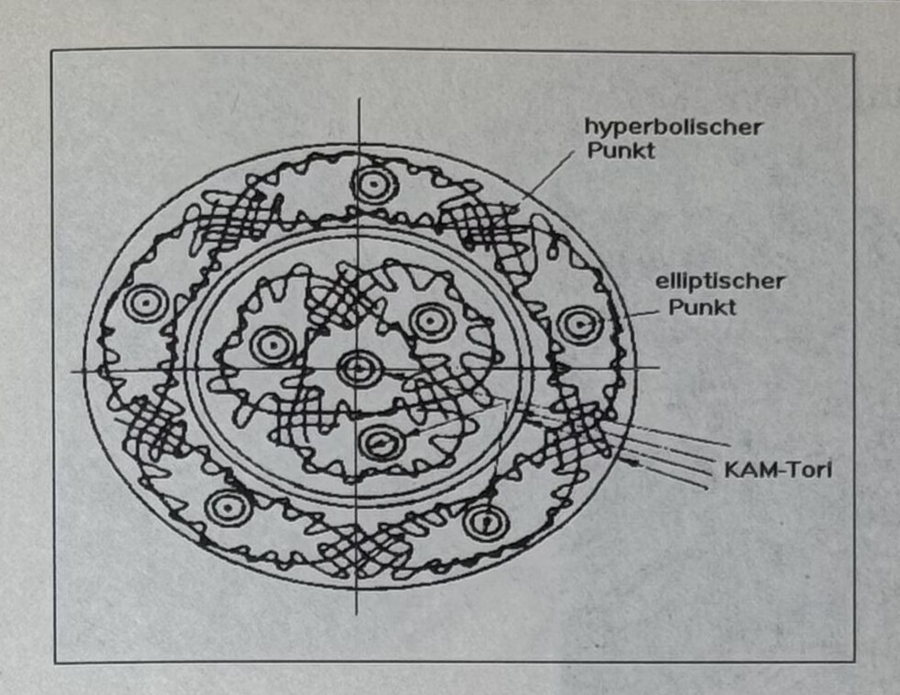


Bild 8.4 Schematischer Schnittes durch einen KAM-Torus nach Birkhoff

Vernachlässigt man in der quadratischen Hénon-Abbildung (Abschnitt 8.1) die Terme x^2, so geht dieses System in die Twist-Map über. Dieses Henon-System und damit das zugehörige Hamilton-System kann somit als Störung des Twist-Systems aufgefaßt werden. Wegen der Störung sind die invarianten Kurven keine exakten Kreise mehr.

Betrachtet man nun zwei Bahnen $(r_0,\theta_0),(r_n,\theta_0)$, so werden die Bahnen i.a. nicht identisch sein. Wegen der kleinen Störung kann das System jedoch als konservativ angesehen werden; d.h. die Fläche zwischen den Bahnen bleibt konstant.

Die Bahnen müssen sich daher schneiden. Es läßt sich zeigen, daß diese Schnittpunkte abwechselnd hyperbolisch und elliptisch sind. Ein hyperbolischer Punkt ist dabei verbunden mit stabilen und unstabilen Mannigfaltigkeiten, ein elliptischer Punkt mit einer stabilen Mannigfaltigkeit in Form eines invarianten Torus. Vergleiche dazu Bild 8.4, das nach einer Skizze von G. D. Birkhoff 1922 (also noch vor dem Computerzeitalter) entworfen wurde.

Wendet man die Theorie auf das quadratische Hénon-System an, so zeigen sich für $a < 1.2$ die ungestörten Kreise der Twist-Map. In der Nähe von $a = 1.3$ entstehen, wegen $\frac{2\pi}{1.3} \approx 5$, fünf Inseln; am Rand erscheinen chaotische Bänder (vgl. Bild 8.1). Wächst a auf 1.58, so ist die Störung so groß, daß alle Kreise zerstört werden. Wegen $\frac{2\pi}{1.58} \approx 4$ erscheint hier eine 4:1 Resonanz (Bild 8.2).

8.5 Standard-Abbildung



Bild 8.5 Standard-Abbildung für $\varepsilon = 1.0$

Die Standard-Abbildung (englisch *standard map*) ist definiert durch

$$x_{n+1} = x_n + y_n + \varepsilon \sin x_n$$
$$y_{n+1} = y_n + \varepsilon \sin x_n$$

Sie wurde von B. V. Schirikow als Poincaré-Abbildung eines periodisch angestoßenen Pendels eingeführt (*Phys. Rep.* 52,81979). Setzt man $\varepsilon = 0$, so hat das entstehende (ungestörte) System

$$x_{n+1} = x_n + y_n$$
$$y_{n+1} = y_n$$

die Jacobi-Determinante eins.

$$\det \mathbf{J} = \begin{vmatrix} 1 & 1 \\ 0 & 1 \end{vmatrix} = 1$$

Das System ist daher für $\varepsilon = 0$ konservativ. Plottet man das Phasendiagramm der Standard-Abbildung, so sieht man für kleine Werte von ε, etwa $\varepsilon = 0.1172$ deutlich die ungestörten Tori, die von $x = 0$ bis $x = 2\pi$ verlaufen. Sie bestehen aus den nicht-resonanten Tori des ungestörten Systems. Steigert man den Parameterwert auf $\varepsilon = 1.0$ bzw. $\varepsilon = 1.8$, so zerfallen die Tori des ungestörten Systems immer mehr. Man vergleiche dazu Bild 8.5 bzw. Farbtafel 15 und Bild 8.6).



Bild 8.6 Standard-Map für $\varepsilon = 1.8$

Es entstehen die Inseln der rationalen Frequenzen. Für $\varepsilon = 4.0$ herrscht Chaos. Hier haben nur die elliptischen Punkte der Periode Eins und der Periode Zwei $(\pi, 0) \leftrightarrow (\pi, \pi)$ Bestand (Bild 8.7).

Die Standard-Abbildung kann mittels des Quick Basic-Programms `standard.bas` ausgeführt werden.

```
'stdrd.bas
'Standard-Abbildung
CONST k = 1!
CONST pi = 3.141529653#: p2 = 2 * pi
DIM i, j AS INTEGER
DIM x, y, x1 AS SINGLE
SCREEN 12: CLS
WINDOW (0, 0)-(p2, p2)
RANDOMIZE TIMER
FOR j = 1 TO 250
  x = p2 * RND: y = p2 * RND
  FOR i = 1 TO 750
    x1 = x + y + k * SIN(x)
    y = y + k * SIN(x)
    x = x1
    IF x > p2 THEN x = x - p2
    IF y > p2 THEN y = y - p2
    IF x < 0 THEN x = x + p2
    IF y < 0 THEN y = y + p2
    PSET (x, y), 1 + j MOD 16
  NEXT i
NEXT j
LOCATE 27, 1: PRINT "Phasendiagramm der Standard Map"
e$ = INPUT$(1): SCREEN 0
END
```



Bild 8.7 Standard-Map für $\varepsilon = 4.0$

8.5 McKay-Abbildung

Die McKay-Abbildung wurde in *Period Doubling as a Universal Route to Stochasticity*, in Long-Time Prediction in Dynamics, (1983) publiziert. Die Abbildung ist gegeben durch

$$x_{n+1} = -y_n + f(x_n)$$
$$y_{n+1} = x_{n+1} - f(x_{n+1})$$

mit der Funktion $f(x) = px - (1-p)x^2$.

Für $|p| < 1$ ist der Ursprung ein elliptischer Fixpunkt. Wird $p < -1$, so verliert der Ursprung seine Stabilität, und es gibt einen Zyklus der Länge 2. Die Periodenverdopplungen wiederholen sich bis zum Punkt

$$p^* = -1.2663117$$

Das System von McKay kann als Störung der Abbildung

$$x \to -y$$
$$y \to x$$

mit der Störfunktion f aufgefaßt werden.

Die Berechnung der Jacobi-Determinante zeigt, daß diese Abbildung konservativ ist:

$$\det \mathbf{J} = \begin{vmatrix} 0 & -1 \\ 1 & 0 \end{vmatrix} = 1$$

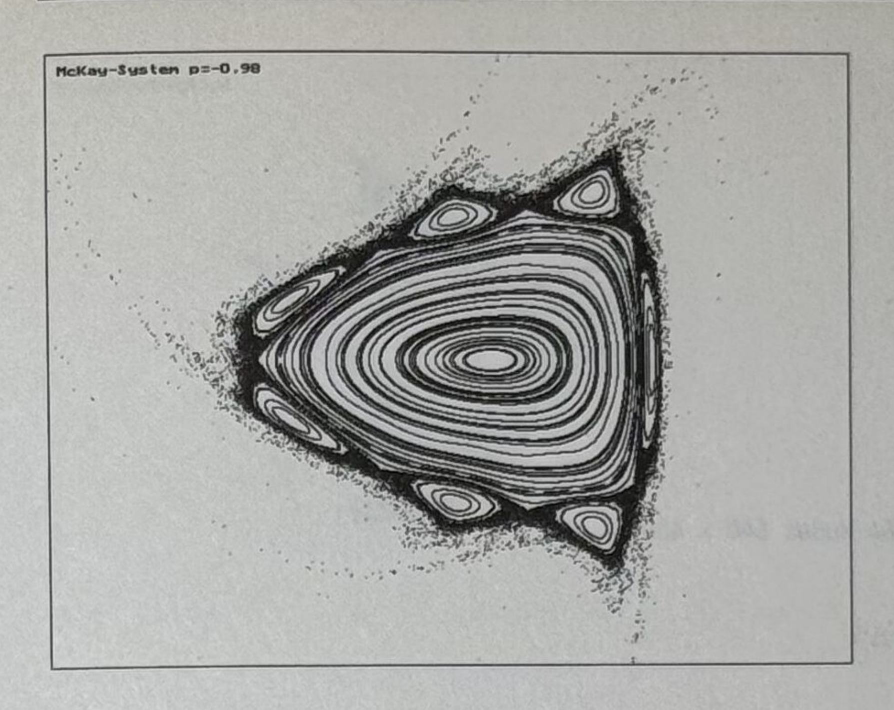


Bild 8.8 McKay-Abbildung für p=-0.98

Für $|p| < 1$ bleiben die Tori der ungestörten Abbildung gemäß Bild 8.8 erhalten. Für $p < -1$ kommt es zur Störung der invarianten Kurven (Bild 8.9), für immer kleiner werdendes p kommt es zum Chaos.

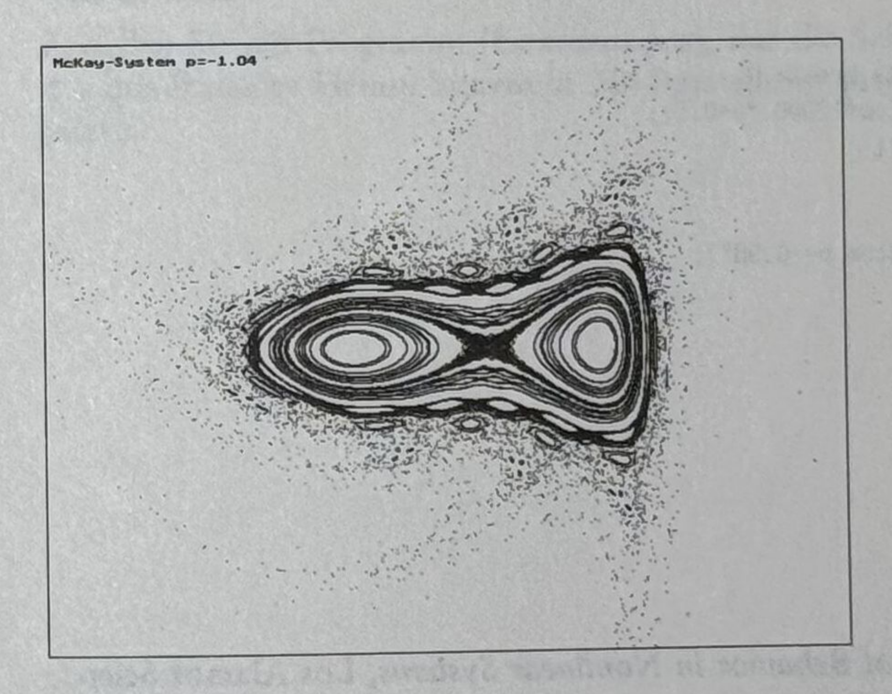


Bild 8.9 McKay-Abbildung für p=-1.04

Die McKay-Abbildung kann mit dem Turbo C-Programm `mckay.c` ausgeführt werden.

```
/* mckay.c */

#include <stdio.h>
#include <graphics.h>
```

```
#include <conio.h>
#include <stdlib.h>
#include <time.h>
#include <math.h>

const double p=-0.98;

void main(void)
{
int a,b,i,j;
double x,y,x1;
time_t now;
int gdriver,gmode;

gdriver = 9; gmode = VGAHI; /* VGA-Modus 640 x 480 */
initgraph(&gdriver,&gmode," ");
rectangle(0,0,639,479);
srand((unsigned) time(&now) % 4001);

for (j=1; j<=75; j++)
{
x = 0.1*rand()/32768.-0.1;
y = 0.1*rand()/32768.-0.1;
for (i=1; i<=12000; i++)
  {
  x1 = -y+p*x-(1-p)*x*x;
  y = x-p*x1+(1-p)*x1*x1;
  x = x1;
  if ((fabs(x)>1) || (fabs(y)>1)) break;
  a = floor(1000.*x+0.5); b = floor(2000.*y+0.5);
  putpixel(a+350,240-b,1+j % 16);
  }
}
moveto(10,10);outtext("McKay-System p=-0.98");
do {} while(!(kbhit()));
closegraph();
return;
}
```

8.6 Ergänzungen

Übung 8.6.1

Von M. Feigenbaum (*Universal Behavior in Nonlinear Systems*, Los Alamos Science 1, 1980) stammt das System

$$x_{n+1} = y_n - x_n^2$$
$$y_{n+1} = a + bx_n$$

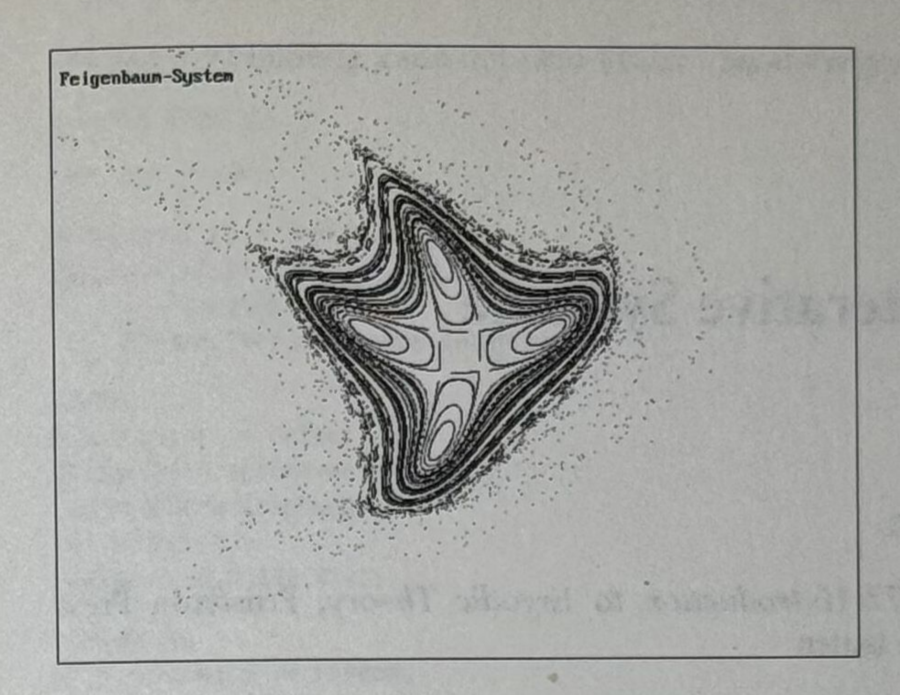


Bild 8.10 Feigenbaum-System für a=0.1,b=-1

Das System ist für $b = -1$ konservativ. Realisieren Sie die Abbildung für $a = 0.1; b = -1$ mittels eines Computer-Programms (*feigsys.bas*).

Übung 8.6.2

Erstellen Sie ein Programm (*kamtorus.pas*), das die Schnitte durch den KAM-Torus des quadratische Hénon-System in 3D-Darstellung ausgibt. Das Ergebnis zeigt Farbtafel 5.



9 Iterative Systeme

9.1 Sinai-Gleichungen

Die von Yaschi G. Sinai 1972 (*Introduction to Ergodic Theory*, Princeton Press 1972) angebenen Gleichungen lauten

$$x_{n+1} = x_n + y_n + \frac{g}{2\pi}\cos(2\pi y_n) \quad \text{mod}\,1$$
$$y_{n+1} = x_n + 2y_n \quad \text{mod}\,1$$

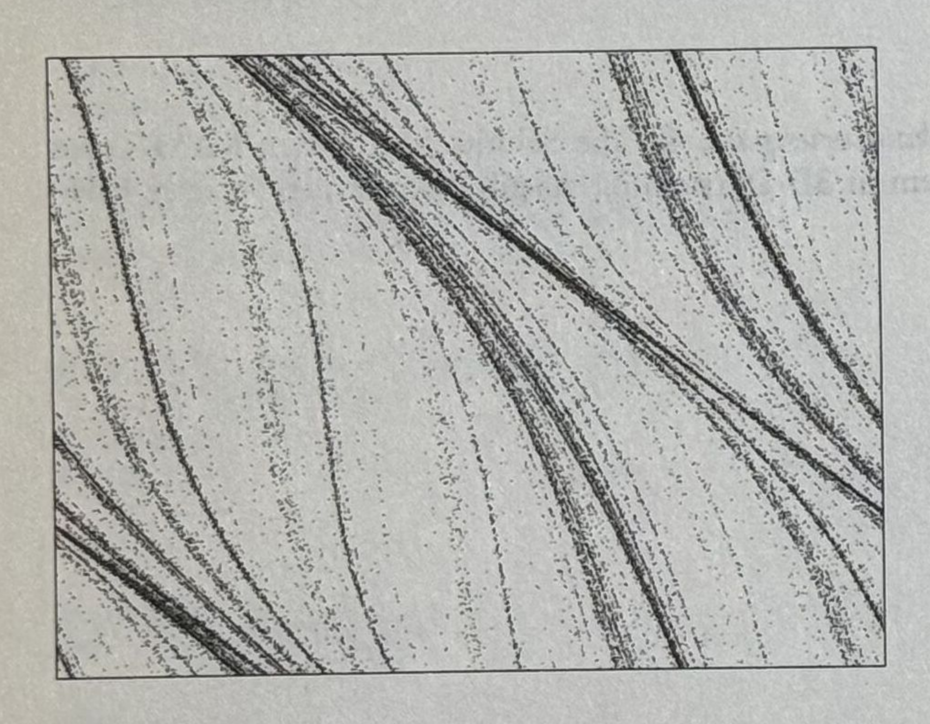


Bild 9.1 Phasen-Plot des Sinai-Systems mit g=1

Sinai konnte zeigen, daß für kleine Werte von g jeder Punkt des Einheitsquadrats mit gleicher Häufigkeit aufgesucht wird. Somit stellt das ganze Quadrat $\lfloor 0;1 \rfloor \times \lfloor 0;1 \rfloor$ einen Attraktor dar. Der Phasenplot des Sinai-Systems hat Ähnlichkeit mit der Faserstruktur von Holz. Das Sinai-System kann als Störung der im nächsten Abschnitt behandelten Katzen-Abbildung aufgefaßt werden.

Die Sinai-Abbildung kann mit dem Turbo Pascal-Programm `sinai.pas` erzeugt werden.

```
program sinai;

uses crt,graph;

const g=1; p2 = 2*pi;
var  x,y,x1,y1: real;
     i : longint;
     Graphdriver,Graphmode:integer;

begin
GraphDriver := Detect;
Initgraph(GraphDriver,GraphMode,' ');
SetgraphMode(Graphmode);
setbkcolor(6);
rectangle(0,0,639,479);

randomize;
x := random; y := random;
for i:=1 to 50000 do
   begin
   x1 := x+y+g*cos(p2*y)/p2;
   y1 := x+2*y;
   x := frac(x1); y := frac(y1);
   putpixel(round(640*x),round(480*y),14);
   end;
repeat until keypressed;
closegraph;
textmode(lastmode)
end.
```

9.2 Katzen-Abbildung

Die Sinai-Gleichungen gehen für $g=0$ in die sog. Katzen-Abbildung (englisch *cat map*) über

$$\left.\begin{aligned} x_{n+1} &= x_n + y_n \\ y_{n+1} &= x_n + 2y_n \end{aligned}\right\} \bmod 1$$

Das System wurde 1967 von D. V. Anosow publiziert, ist aber erst 1968 populär geworden durch die Interpretation von Arnold und Avez als Katzen-Abbildung. Diese Abbildung gab dem System den Namen (Bild 9.2). Das System ist konservativ wegen der Jacobi-Determinante

$$\det \mathbf{J} = \begin{vmatrix} 1 & 1 \\ 1 & 2 \end{vmatrix} = 1$$

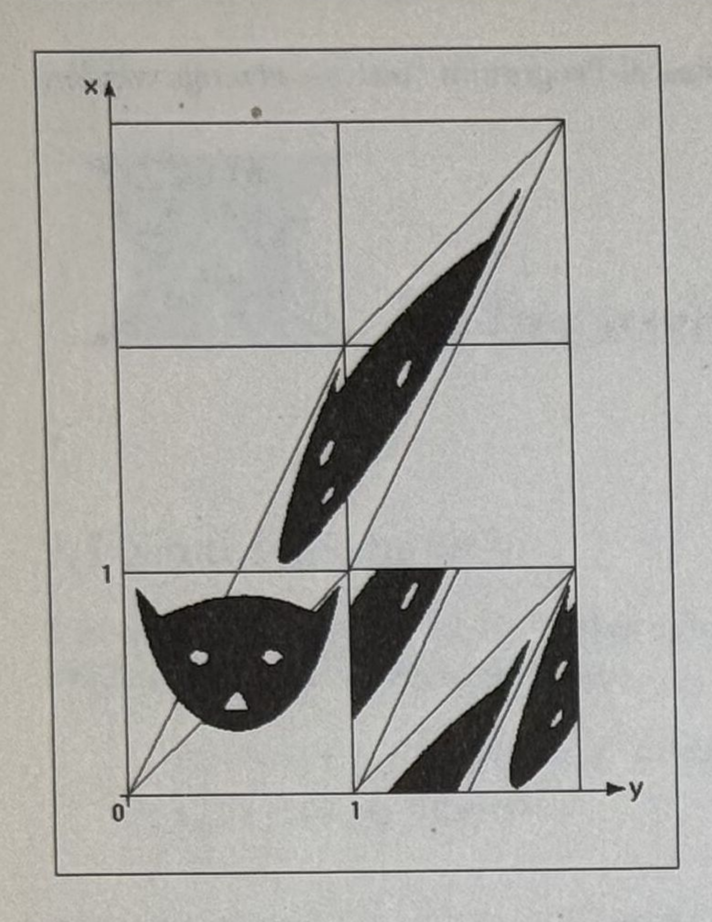


Bild 9.2 Katzen-Abbildung (Cat Map)

Die Eigenwerte ergeben sich aus der charakterischen Gleichung

$$\det(\mathbf{J} - \mu\mathbf{E}) = \mu^2 - 3\mu + 1 \text{ zu } \mu_1 = \frac{3+\sqrt{5}}{2}; \quad \mu_2 = \frac{1}{\mu_1}$$

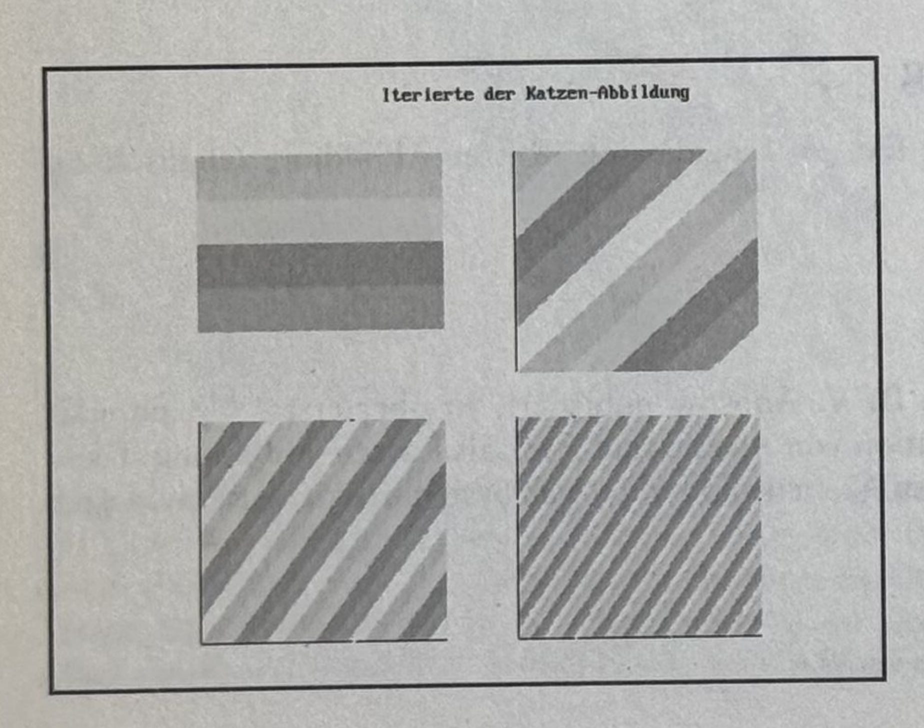


Bild 9.3 Iterierte der Katzen-Abbildung

Die Ljapunow-Exponenten sind die zugehörigen Logarithmen

$$\lambda_1 = 0.962; \quad \lambda_2 = -0.962$$

Der Ursprung stellt einen Sattelpunkt dar; d.h. er ist hyperbolisch. Wie man dem Bild 9.3 entnimmt, wird das Einheitsquadrat bei der iterierten Abbildung gestaucht und gefaltet. Dies ist ein typischer Weg ins Chaos; wie z.B. ein Farbklecks sich beim Umrühren verhält.

Die Graphik kann mit dem Quick Basic-Programm `cat.bas` erzeugt werden.

```
'cat.bas
'Die ersten 3 Iterierten der Katzen-Abbildung

DIM x, x1, x2, x3, y, y1, y2, y3 AS SINGLE
DIM c AS INTEGER

SCREEN 12: CLS
WINDOW (0, 0)-(3.4, 2.8)
LINE (.6, .2)-(1.6, 1.2), , B
LINE (.6, 1.4)-(1.6, 2.4), , B
LINE (1.9, 1.4)-(2.9, 2.4), , B
LINE (1.9, .2)-(2.9, 1.2), , B
LOCATE 2, 33: PRINT "Cat-Map"

FOR x = 0 TO 1 STEP .005
FOR y = 0 TO 1 STEP .005
   c = INT(y * 5) + 1
   PSET (x + .6, y + 1.4), c
   x1 = x + y
   y1 = x + 2 * y
   x1 = x1 - INT(x1): y1 = y1 - INT(y1)
   PSET (x1 + 1.9, y1 + 1.4), c
   x2 = x1 + y1
   y2 = x1 + 2 * y1
   x2 = x2 - INT(x2): y2 = y2 - INT(y2)
   PSET (x2 + .6, y2 + .2), c
   x3 = x2 + y2
   y3 = x2 + 2 * y2
   x3 = x3 - INT(x3): y3 = y3 - INT(y3)
   PSET (x3 + 1.9, y3 + .2), c
NEXT y
NEXT x
e$ = INPUT$(1): SCREEN 0
END
```

9.3 Bäcker-Abbildung

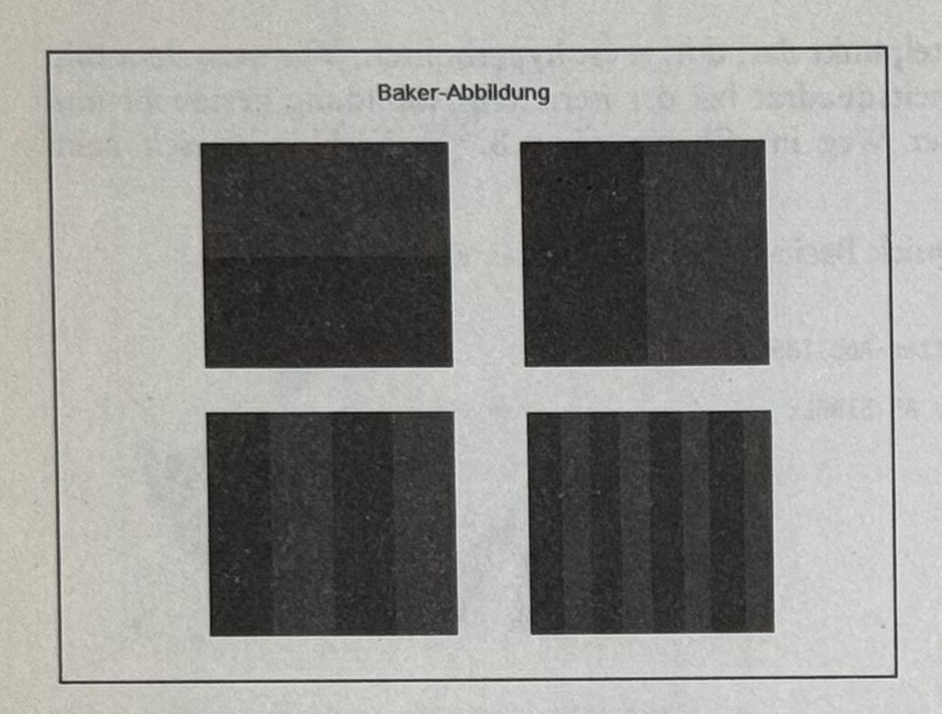


Bild 9.4 Die ersten 3 Iterierten der Bäcker-Abbildung

Die Bäcker-Abbildung (englisch *baker map*) liefert einen ähnlichen Weg ins Chaos wie die Katzenabbildung. Sie ist benannt nach der Tätigkeit eines Bäckers, der den Teig in ähnlicher Weise breitwalzt und faltet. Die Gleichung ist gegeben durch die Zuordnung

$$x_{n+1} = \begin{cases} 2x_n & \textit{für } 0 \le x_n \le 0.5 \\ 2x_n - 1 & \textit{für } 0.5 < x_n \le 1 \end{cases}$$

$$y_{n+1} = \begin{cases} 0.5y_n & \textit{für } 0 \le x_n \le 0.5 \\ 0.5(y_n + 1) & \textit{für } 0.5 < x_n \le 1 \end{cases}$$

Wie man sieht, wird das Einheitsquadrat in x-Richtung um den Faktor 2 gestreckt und in y-Richtung um den Faktor 2 gestaucht. Durch die Modulo 1-Bildung wird das Ergebnis wieder auf das Einheitsquadrat abgebildet.

9.4 Hufeisen-Abbildung

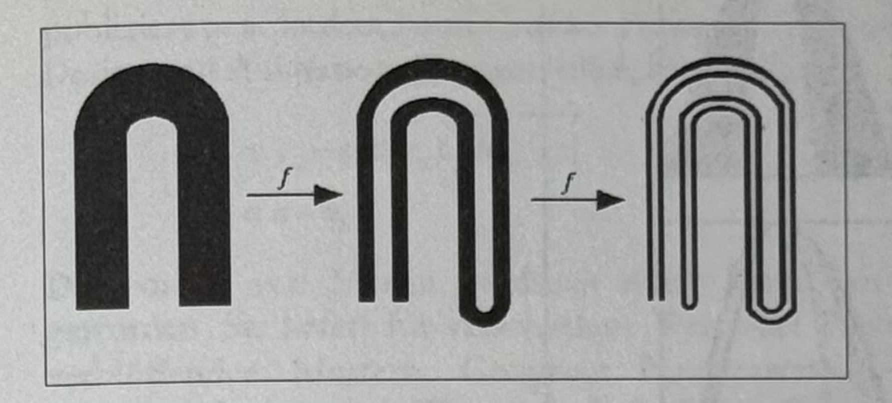


Bild 9.5 Iterierte der Hufeisen-Abbildung

Die Hufeisen-Abbildung (englisch *horse shoe map*) wurde 1967 von S. Smale als Weg in das Chaos erdacht, um als Phasenraum-Modell des Van der Pol-Attraktors zu dienen. Ähnlich wie bei der Bäcker-Abbildung, wird hier das Einheitsquadrat in x-Richtung um den Faktor 2 gestaucht und in y-Richtung um den Faktor 2 gestreckt. Sodann wird der dünne Streifen in der Mitte abgeknickt und zu einem Hufeisen gebogen.

Ursprünglich hatte Smale gehofft, alle dynamischen Prozesse durch Dehnen und Stauchen beschreiben zu können. Zu seiner Enttäuschung zeigte es sich jedoch, daß auf den Prozeß des Faltens nicht verzichtet werden konnte.

Für seine Abbildung gab er keine explizite Formel an; jedoch liefert die von ihm geschaffene symbolische Dynamik den notwendigen Formalismus. Eine Darstellung der symbolischen Dynamik ist im Rahmen des Buches nicht möglich; eine schöne Einführung findet man bei Devaney [06].

Eine konkrete Hufeisenabbildung wurde von H. Lauwerier in *A case of a not so strange strange attractor*, Report Amst. Centre Math. Comp. Sci. AM-R8402 (1984) angegeben. Das Einheitsquadrat wird damit auf ein hufeisenförmiges Gebilde innerhalb des Quadrats abgebildet. Die Abbildung lautet

$$x_{n+1} = \frac{1}{3} x_n (1 - 2y_n) + y_n$$
$$y_{n+1} = 4 y_n (1 - y_n)$$

Wie man sieht, kann das System als Verallgemeinerung der logistischen Gleichung aufgefaßt werden. Die ersten 3 Iterierten der Lauwerier-Abbildung können dem Bild 9.6 entnommen werden.

Die Bildschirmabbildung kann mit dem Quick Basic-Programm `hufeis.bas` erstellt werden.

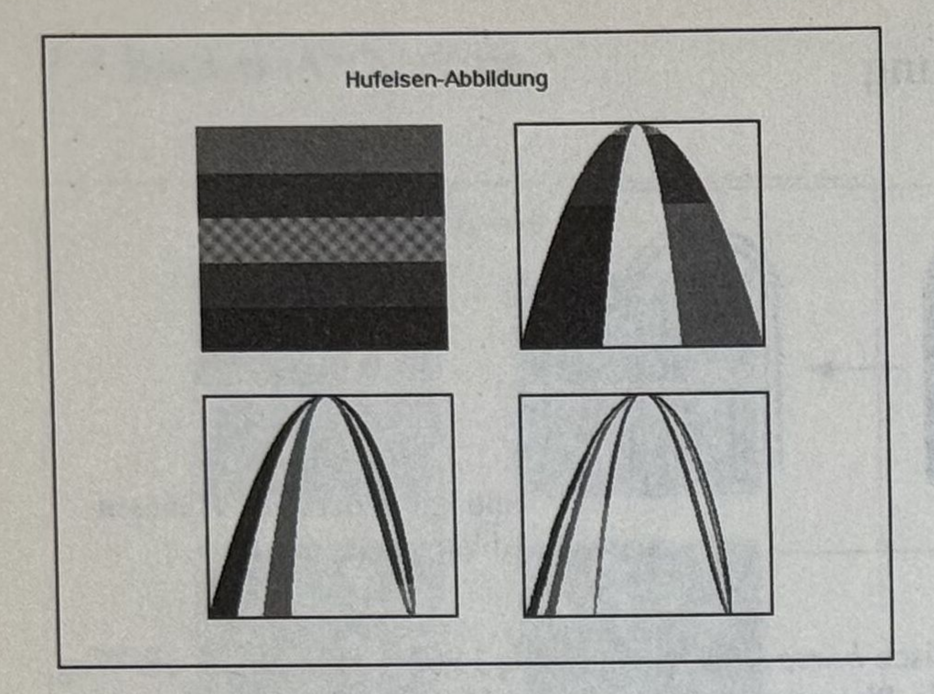


Bild 9.6 Iterierte der Lauwerier-Hufeisen-Abbildung

```
'hufeis.bas
'Hufeisen-Abbildung von Lauwrier

DIM x, x1, x2, x3, y, y1, y2, y3 AS SINGLE
DIM c AS INTEGER
SCREEN 12: CLS
WINDOW (0, 0)-(3.4, 2.8)
LINE (.6, .2)-(1.6, 1.2), , B
LINE (.6, 1.4)-(1.6, 2.4), , B
LINE (1.9, 1.4)-(2.9, 2.4), , B
LINE (1.9, .2)-(2.9, 1.2), , B
LOCATE 2, 23: PRINT "Hufeisen-Abbildung und Iterierte"

FOR x = 0 TO 1 STEP .001
FOR y = 0 TO 1 STEP .001
    c = INT(y * 5) + 1
    PSET (x + .6, y + 1.4), c
    x1 = x / 3 * (1 - 2 * y) + y: '1.Iterierte
    y1 = 4 * y * (1 - y)
    PSET (x1 + 1.9, y1 + 1.4), c
    x2 = x1 / 3 * (1 - 2 * y1) + y1: '2.Iterierte
    y2 = 4 * y1 * (1 - y1)
    PSET (x2 + .6, y2 + .2), c
    x3 = x2 / 3 * (1 - 2 * y2) + y2: '3.Iterierte
    y3 = 4 * y2 * (1 - y2)
    PSET (x3 + 1.9, y3 + .2), c
NEXT y
NEXT x
e$ = INPUT$(1): SCREEN 0
END
```

9.5 Martin-Abbildung

Die von B. Martin von der Aston University (Birmingham) angegebene Abbildung, publiziert u. a. in dem Band *Graphic Potential of Recursive Functions*, in Computers, Design and Animation, Springer 1989, hat die Form

$$x_{n+1} = y_n - \operatorname{sgn}(x_n)\sqrt{|bx_n - c|}$$
$$y_{n+1} = a - x_n$$

Die Formel von Martin ist durch einen Artikel im Scientific American bekannt geworden. Sie liefert für verschiedene Werte der Parameter a,b,c eine Vielzahl von verblüffenden Mustern. Geeignete Parameterwerte finden sich in: *Computer-Kurzweil*, Spektrum d. Wissenschaft, 9 (1986). Ein schönes teppichähnliches Muster erhält man, z.B., für $a = 200$, $b = 0.1$, $c = -80$ (siehe Bild 9.7 und Farbtafel 7).

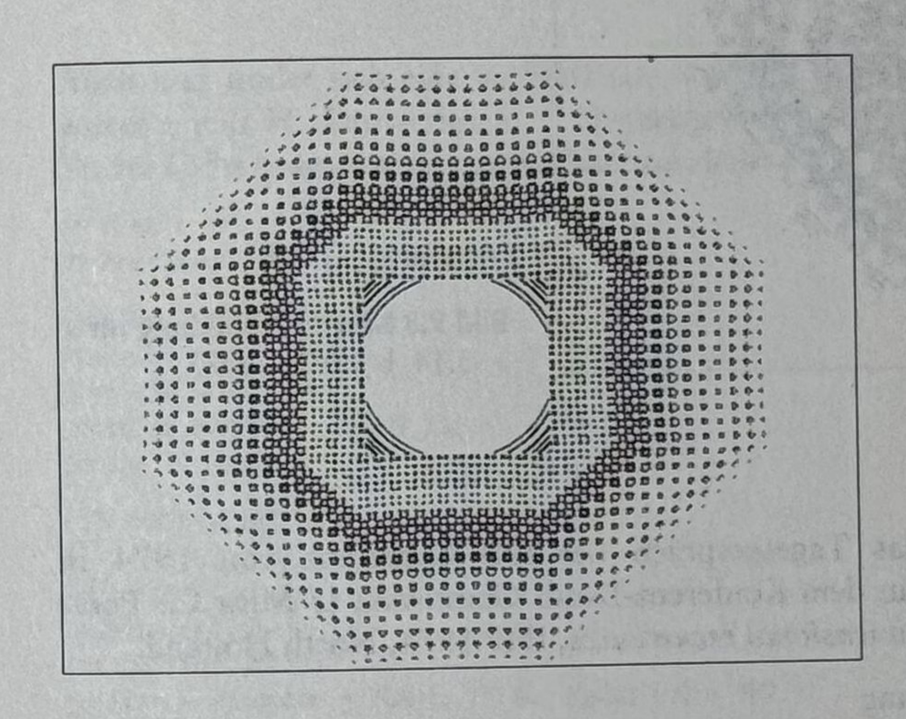


Bild 9.7 Martin-Abbildung für a=-200, b=0.1, c = - 80

Die Abbildungen wurden mit dem Basic-Programm `martin.bas` erstellt.

```
'martin.bas

CONST a = -200: b = .1: c = -80: 'Parameter
DIM x, y, x1 AS DOUBLE
DIM i AS LONG

SCREEN 12: CLS
WINDOW (-510, -440)-(330, 240)
LINE (-510, -440)-(330, 240), , B

x = .4: y = .3
```

```
FOR i = 1 TO 140000
     x1 = y - SGN(x) * SQR(ABS(b * x - c))
     y = a - x
     x = x1
     PSET (x, y), 1 + i \ 10000
NEXT i
e$ = INPUT$(1): SCREEN 0
END
```

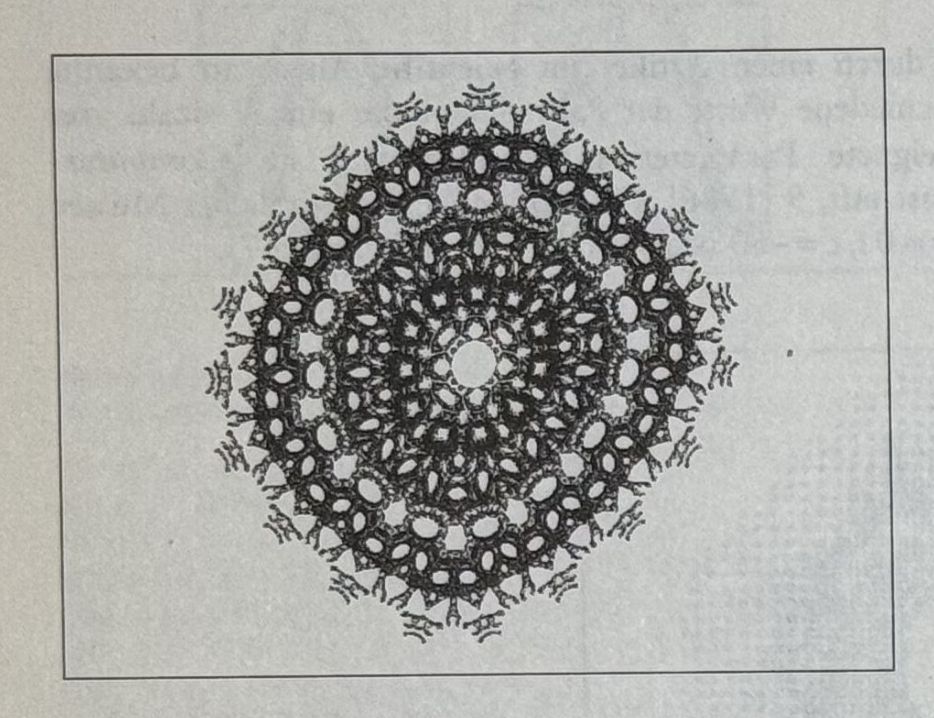


Bild 9.8 Martin-Abbildung für a = -3.14, b = 0.3, c = 0.3

9.6 Mira-Abbildung

Die Mira-Abbildung war das Tagesgespräch der IFIP-Conferenz von 1974 in Stockholm. Sie ist publiziert in dem Konferenz-Band: Gumowski I./ Mira C.: *Point sequences generated by two-dimensional recurrences*, IFIP 1974, North Holland.

Die Abbildung hat die Gleichung

$$x_{n+1} = by_n + F(x_n)$$
$$y_{n+1} = -x_n + F(x_{n+1})$$

Dabei ist F die Funktion

$$F(x) = ax - (1-a)\frac{2x^2}{1+x^2}$$

Tafel 3: Magnetisierungszonen des Ising-Modells nach Oliviera

Tafel 1: Ljapunow-Diagramm der Kreisgleichung

Tafel 2: Ljapunow-Diagramm der Gleichung $x = 2.5\sin^2(x+r)$

Tafel 7: Martin-Abbildung für a=-200,b=0.1,c=-80

Tafel 5: 3D-Darstellung eines KAM-Torus

Tafel 6: 3D-Farnblätter nach Barnsley

Tafel 11: Alternierender Phythagoras-Baum

Tafel 9: Julia-Mengen der Gleichung z^3-1=0

Tafel 10: Quadr.Hénon-Abbildung für a=1.35

Tafel 15: Phasenplot der Standard-Abbildung für eps=1

Tafel 13: Mandelbrot-Menge der Gleichung f(z)=c+z+exp(z)

Tafel 14: Riemann-Projektion von Tafel 9 auf Kugel

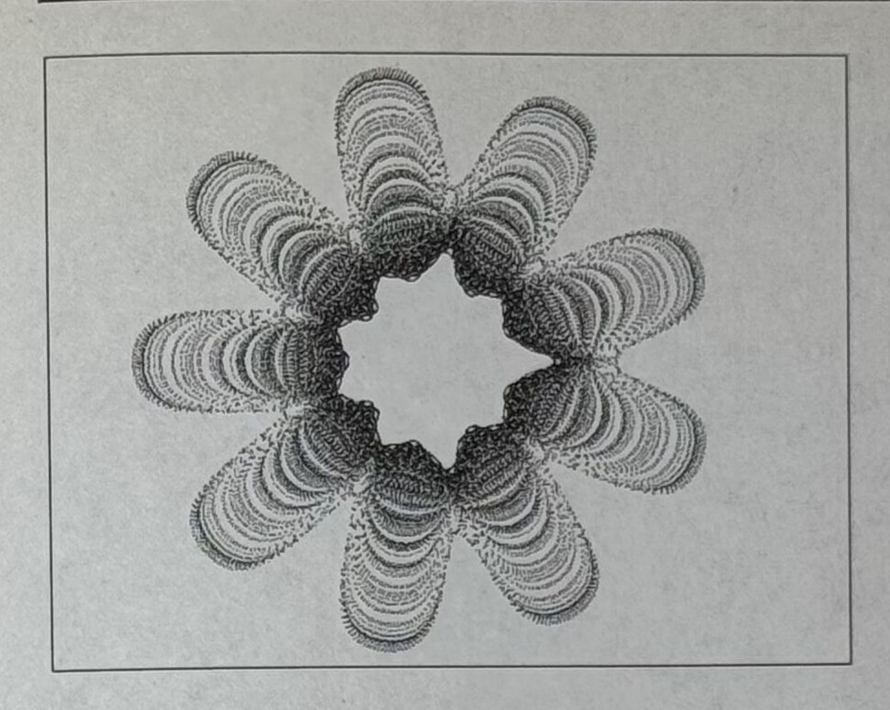


Bild 9.9 Mira-Abbildung für a=0.4, b=1.0

Auch hier findet sich eine verblüffende Vielzahl von Mustern. Gute Parameterwerte wurden von H. Lauwerier in [12] angegeben. Die Mira-Abbildung kann mit dem Turbo C-Programm `mira.c` ausgeführt werden.

```
/* mira.c */
/* Abbildung von Mira & Gumowski */

#include <stdio.h>
#include <graphics.h>
#include <conio.h>
const double a=-0.48,b=0.93;
double f(double);

void main(void)
{
long int i;
double x,x1,y;
int gdriver,gmode;
gdriver = 9; gmode = VGAHI; /* VGA-Modus 640 x 480 */
initgraph(&gdriver,&gmode," ");
rectangle(0,0,639,479);

x = 4.; y = 0.;
for (i=1; i<=120000L; i++)
  {
  x1 = b*y + f(x);
  y = -x + f(x1);
  x = x1;
  putpixel(350+(int)(x*26.0),280-(int)(y*26.0), 1+(int) (i/10000));
  }
do {} while(!(kbhit()));
closegraph();
```

```
return;
}

double f(double x)
{
return a*x-(1.-a)*2.*x*x/(1.+x*x);
}
```

9.7 Pickover-Abbildung

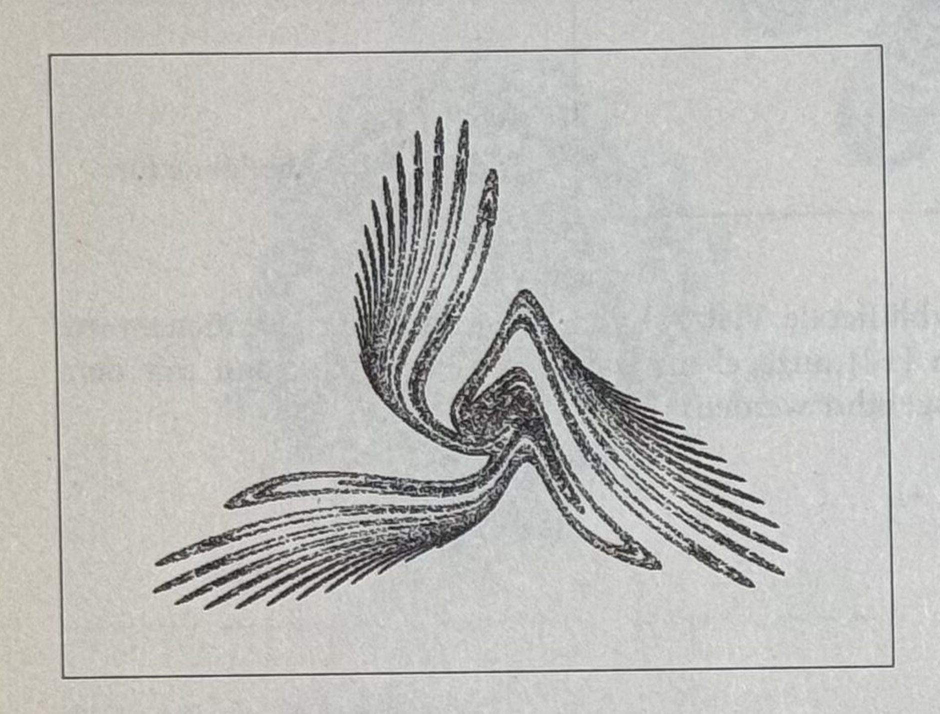


Bild 9.10 Mira-Abbildung für a=-0.4.8, b=0.93

Die Pickover-Abbildung ist durch Diskretisierung einer partiellen Differentialgleichung entstanden. Wegen ihrer orginellen Form erhielt sie den Namen *Popcorn.* Das System lautet

$$x_{n+1} = x_n - h(\sin y_n + \tan(3y_n))$$
$$y_{n+1} = y_n - h(\sin x_n + \tan(3x_n))$$

Die Größe des Popcorn-Gitters ist durch den Diskretisierungsparameter h des Differentialgleichungs-Systems gegeben. Realisieren Sie die Graphik für $h = 0.05$ als Übung 9.8.1 (vgl. Bild 9.11). Die Abbildung ist ebenfalls durch einen Artikel im Scientific American bekannt geworden.

9.8 Ergänzungen und Anregungen

Es gibt eine Vielzahl von Iterativen Systemen, die interessante Muster oder Attraktoren erzeugen. Einige davon sollen als Übungen erstellt werden.

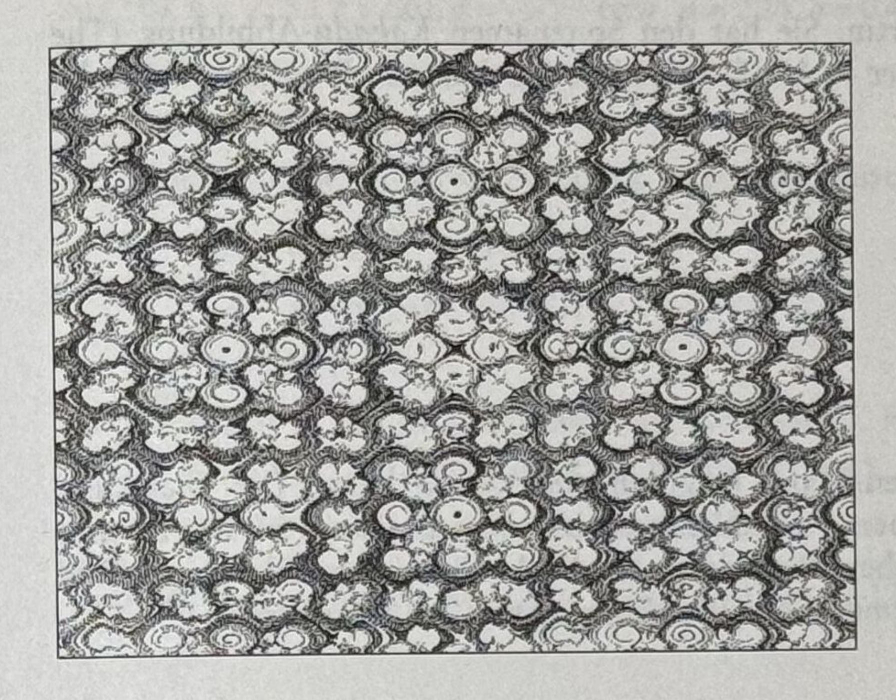


Bild 9.11 Popcorn von Pickover

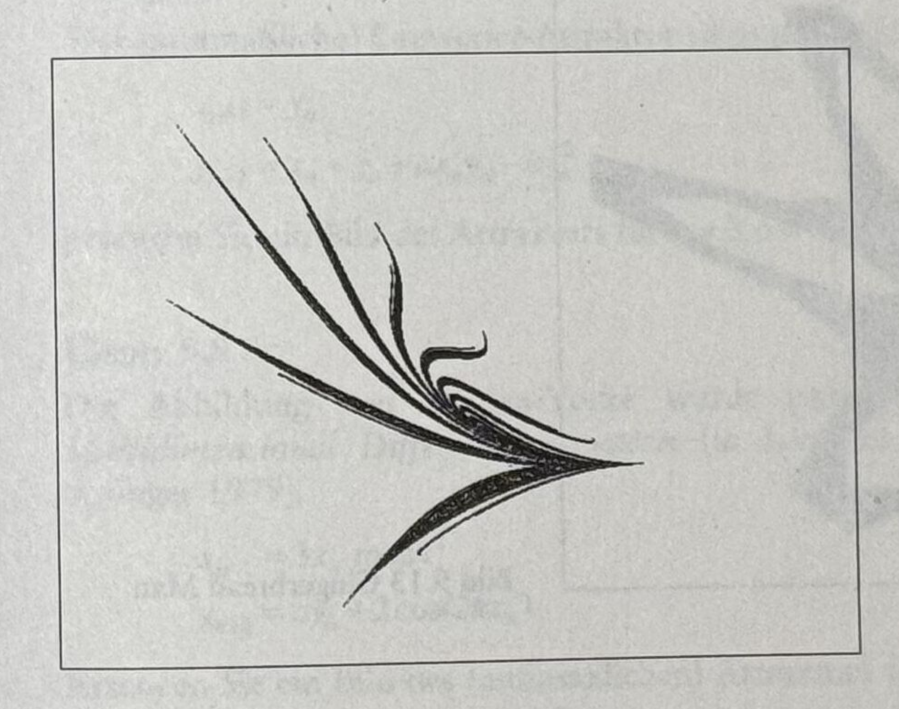


Bild 9.12 Kakadu-Abbildung mit $a = 0.7$; $b = 1.2$; $c = 0.21$

Übung 9.8.1

Realisieren Sie die Popcorn-Abbildung von Clifford Pickover am Rechner (Programm *popcorn*).

Übung 9.8.2

Die Abbildung

$$x_{n+1} = y_n(1+\sin(ax_n)) - b\sqrt{|x_n|}$$

$$y_{n+1} = c - x_n$$

stammt ebenfalls von B. Martin. Sie hat den Spitznamen *Kakadu*-Abbildung (The *cockatoo*, in Math.Intelligencer 9, 1987). Erstellen Sie dazu ein Programm *kakadu*! (Bild 9.12)

Übung 9.8.3

Die Abbildung

$$x_{n+1} = 1 - y_n + |x_n|$$

$$y_{n+1} = x_n$$

erzeugt den sog. Pfefferkuchen-Mann (englisch *gingerbread man)*. Die Gleichung wurde von L. Devaney in seinem Artikel *Chaotic dynamical systems* in The Science of Fractal Images [23], angegeben. Realisieren Sie die Abbildung am Bildschirm (Bild 9.13). Diese Abbildung ist besonders sensitiv abhängig von den Startwerten.

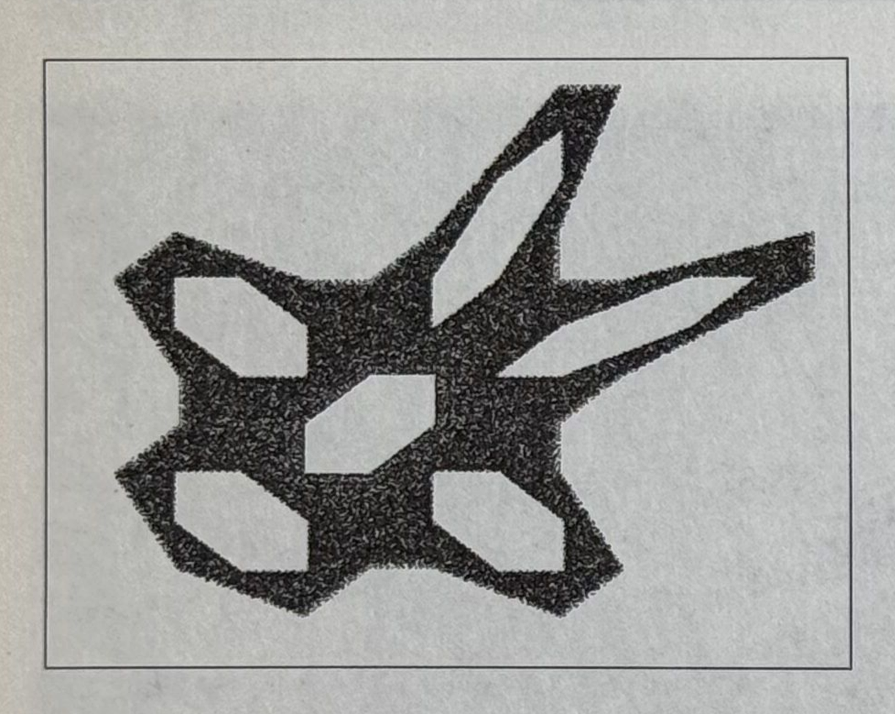


Bild 9.13 Gingerbread Man

Übung 9.8.4

Die Abbildung

$$x_{n+1} = \sin(ay_n) - \cos(bx_n)$$

$$y_{n+1} = \sin(cx_n) - \cos(dy_n)$$

stammt von P. de Jong und ist ein Spezialfall des unten beschriebenen Pickover-Attraktors. Die Formel stammt aus dem Artikel *Computer-Kurzweil*, Spektrum d. Wissenschaft 10, (1987). Realisieren Sie die Graphik für die Parametermengen (Programm *jong*)

$$a = 2.01, b = -2.53, c = 1.61, d = -0.33 \text{ bzw. } a = 2.24, b = 0.43, c = -0.65, d = -2.43.$$

Übung 9.8.5

Der (mutmaßliche) Pickover-Attraktor ist gegeben durch das System

$$\begin{aligned} x_{n+1} &= \sin(ay_n) - z_n \cos(bx_n) \\ y_{n+1} &= z_n \sin(cx_n) - \cos(dy_n) \\ z_{n+1} &= \sin x_n \end{aligned}$$

Es findet sich in Pickovers erstem Buch *Computers, Pattern, Chaos and Beauty*, St. Martin´s Press (1990). Wählen Sie die Parameterwerte

$$a = 2.24, b = 0.43, c = -0.65, d = -2.43$$

und projizieren Sie den Attraktor in die xy- bzw. yz-Ebene (Programm *pickovr*).

Übung 9.8.6

Der (mutmaßliche) Lauwerier-Attraktor ist gegeben durch die Gleichungen

$$\begin{aligned} x_{n+1} &= y_n \\ y_{n+1} &= x_n - y_n + ax_n y_n - by_n^2 \end{aligned}$$

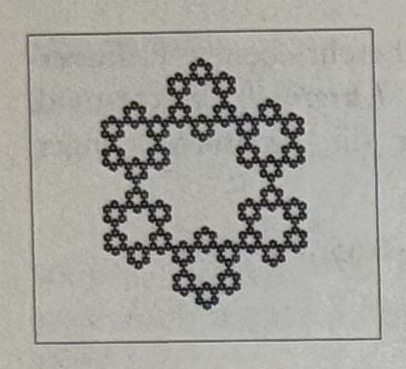
Erzeugen Sie ein Bild des Attraktors für $a = 3, b = 2$ (Programm *lauwer*).

Übung 9.8.7

Die Abbildung von Kaplan-Yorke wurde publiziert in *Chaotic Behavior of Multidimensional Difference Equation* (in Lectures Notes in Mathematics 730, Springer 1979).

$$\begin{aligned} x_{n+1} &= 3x_n \bmod 1 \\ y_{n+1} &= ay_n + 2\cos(2\pi x_n) \end{aligned}$$

Erzeugen Sie ein Bild des (mutmaßlichen) Attraktors für $a = 0.25$. Der Attraktor liegt für $|a| < 1$ in dem Bereich $|y| \le \frac{2}{1-|a|}$ (Programm *kaplan*).



10 Das Chaosspiel

Das von M.Barnsley gefundene Chaosspiel soll an einigen Beispielen erklärt werden.

10.1 Sierpinski-Dreieck

Das Chaosspiel für das Sierpinski-Dreieck funktioniert folgendermaßen. Zuerst wählt man die Eckpunkte A,B,C des Dreiecks geeignet. Dann startet man mit einem beliebigen Punkt x_1 des Bildschirms. Als nächstes wird ein zufälliger Eckpunkt des Dreiecks gewählt; z.B. C. Der Mittelpunkt der Strecke von x_1 nach C wird am Bildschirm als Punkt x_2 markiert. Erneut wird ein neuer Eckpunkt, z.B. B ausgelost. Der Mittelpunkt der Strecke von x_2 nach B wird ebenfalls als Punk x_3 markiert. Das Verfahren setzt sich in der angegebenen Weise fort: Jeweils vom zuletzt markierten Punkt wird der Mittelpunkt der Strecke zu einem zufällig ausgewählten Eckpunkt markiert. Der Vorgang endet, wenn eine genügend große Zahl von Punkten gesetzt ist. Die Gesamtheit aller gesetzten Punkte bildet dann die gesuchte Punktemenge; bei diesem Beispiel entsteht ein Sierpinski-Dreieck.

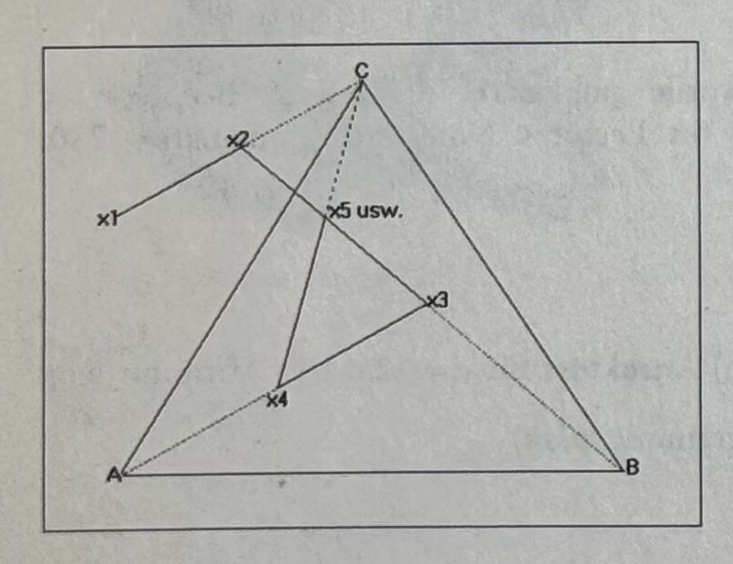


Bild 10.1 Chaosspiel

Das Turbo C-Programm `sierp.c` führt das Chaosspiel für das Sierpinski-Dreieck aus. Die Koordinaten des Dreiecks sind im Programm für VGA-Auflösung so vorgegeben, daß die Spitze oben liegt.

```
/* sierp.c */
/* Erzeugung des Sierpinski-Dreiecks mittels Chaos-Spiel */

#include <stdio.h>
#include <math.h>
#include <graphics.h>
#include <conio.h>
#include <stdlib.h>
#include <time.h>

void main(void)
{
long int i;
int p,x,y;
time_t now;
int gdriver,gmode;
gdriver = 9; gmode = VGAHI; /* VGA-Modus 640 x 480 */
initgraph(&gdriver,&gmode," ");
srand((unsigned) time(&now) % 4001);

x = y = 0;
for (i=1L; i<=120000L; i++)
{
p = rand() % 3;
switch(p)
  {
  case 0: x /= 2; y = (y+479)/2; break;
  case 1: x = (x+320)/2; y /= 2; break;
  case 2: x = (x+639)/2; y = (y+479)/2;
  }
if (i>20) putpixel(x,y,2*p+10);
}
moveto(1,3);outtext("Sierpinski-Dreieck");
moveto(1,13);outtext("mittels Chaos-Spiel");
do {} while(!(kbhit()));
closegraph();
return;
}
```

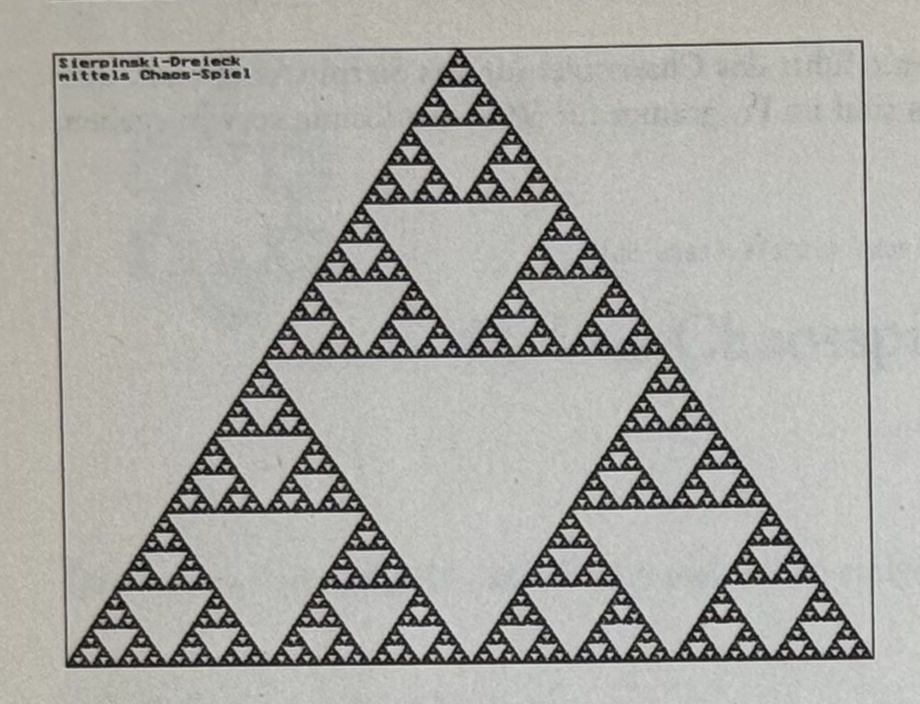


Bild 10.2 Mittels Chaosspiel erzeugtes Sierpinski-Dreieck

10.2 Menger-Teppich

Das Chaosspiel für den Mengerteppich, auch Sierpinski-Teppich genannt, ist etwas komplizierter, da hier keine rotationssymmetrische Figur vorliegt. Da das mittlere ausgeschnittene Quadrat insgesamt 8 Nachbarn hat, muß der Übergang zum ausgelosten Nachbarquadrat koordinatenmäßig vorgegeben werden. Hinzu kommt noch die Stauchung auf $\frac{1}{3}$ der Quadratlänge. Das jeweils besuchte Quadrat wird im Programm mittels Zufallszahl ausgewählt.

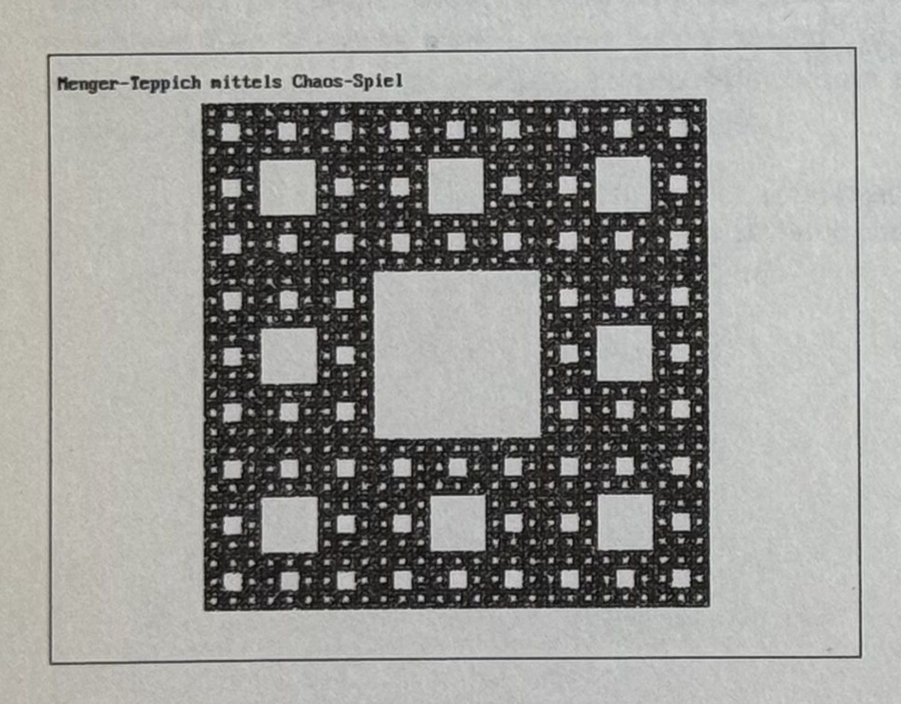


Bild 10.3 Menger-Teppich

Das folgende Quick Basic-Programm `menger.bas` erzeugt mittels Chaosspiel den Mengerteppich.

```
'menger.bas
'Menger-Teppich mittels Chaos-Spiel

DIM r AS INTEGER
DIM i AS LONG
DIM x, y AS SINGLE

SCREEN 12: CLS
WINDOW (-1.6, -1.2)-(1.6, 1.2)
LINE (-1.6, -1.2)-(1.6, 1.2), , B

RANDOMIZE TIMER
x = 1: y = 1
FOR i = 0 TO 200000
   r = 1 + INT(8 * RND)
   SELECT CASE r
     CASE 1: x = (x + 2) / 3: y = (y + 2) / 3
     CASE 2: x = (x + 2) / 3: y = (y - 2) / 3
     CASE 3: x = (x - 2) / 3: y = (y + 2) / 3
     CASE 4: x = (x - 2) / 3: y = (y - 2) / 3
     CASE 5: x = (x + 2) / 3: y = y / 3
     CASE 6: x = (x - 2) / 3: y = y / 3
     CASE 7: x = x / 3: y = (y + 2) / 3
     CASE 8: x = x / 3: y = (y - 2) / 3
   END SELECT
   PSET (x, y), r + 7
NEXT i
LOCATE 2, 2: PRINT "Menger-Teppich mittels Chaos-Spiel"
e$ = INPUT$(1): SCREEN 0
END
```

10.3 Dürer-Fünfeck

Als Beispiel für eine drehsymmetrische Figur soll das Dürer-Fünfeck mittels Chaosspiel erzeugt werden. Das Dürer-Fünfeck findet sich in dem Werk Dürers *Unterweisung der Messung* von 1525.

Das Chaosspiel funktioniert wie beim Sierpinski-Dreieck. Nur wird hier nicht der Mittelpunkt der Verbindungsstrecke zum ausgelosten Eckpunkt gewählt, sondern der Punkt mit der Entfernung 62.4%.

Das Bild 10.4 wuirde mit Hilfe des Turbo Pascal-Programms `duerer.pas` erzeugt. Die Eckpunkte des inneren Fünfecks sind für die VGA-Auflösung im Programm vorgegeben.

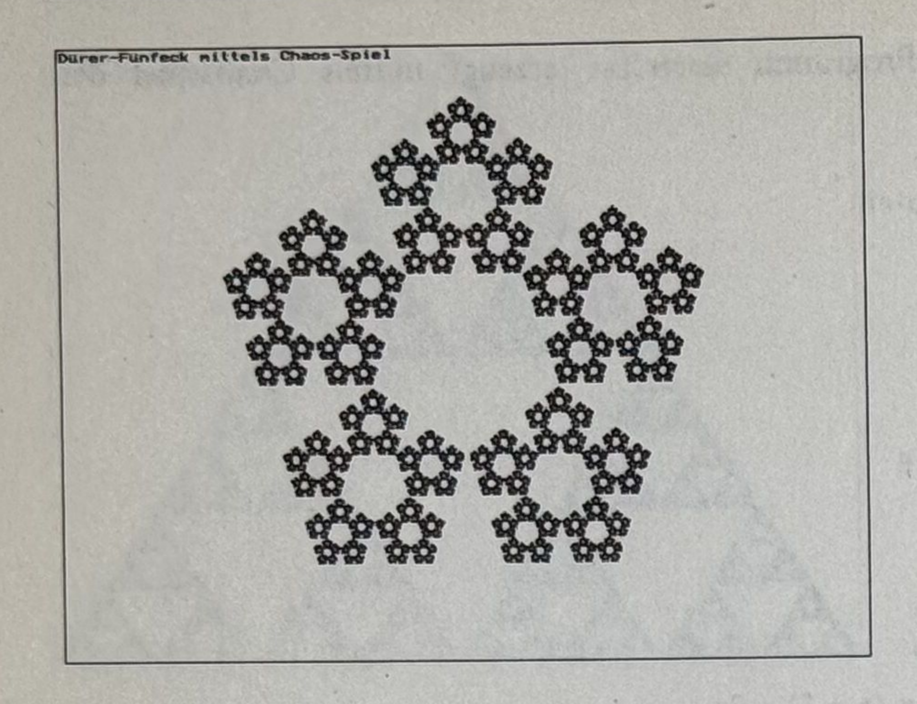


Bild 10.4 Dürer-Fünfeck mittels Chaosspiel.

```
program duerer; { Dürer-Fünfeck mittels Chaos-Spiel }

uses crt,graph;

const N = 0.624;
var p,x,xp,y,yp : integer;
    Graphdriver,Graphmode : integer;
    i : longint;

begin
GraphDriver := 9;
GraphMode := VGAHi; { VGA-Modus 640 x 480 }
Initgraph(GraphDriver,GraphMode,'\tp\bgi');
SetgraphMode(Graphmode);

randomize;
x := 0; y := 0;
for i := 1 to 120000 do

begin
  p := random(5);
  case p of
    0: begin xp := 320; yp := 40 end;
    1: begin xp := 510; yp := 178 end;
    2: begin xp := 438; yp := 402 end;
    3: begin xp := 202; yp := 402 end;
    4: begin xp := 130; yp := 178 end
  end;
  x := x + round((xp-x)*N);
  y := y + round((yp-y)*N);
  if i>20 then putpixel(x,y,p+10);
end;

moveto(3,3);outtext('Dürer-Fünfeck mittels Chaos-Spiel');
```

```
repeat until keypressed;
closegraph;
textmode(lastmode)
end.
```

10.4 Drehsymmetrisches Sechseck

Diese Variante des Chaosspiels liefert drehsymmetrische Sechsecke, die im Inneren die Schneeflockenkurve von Koch begrenzen.

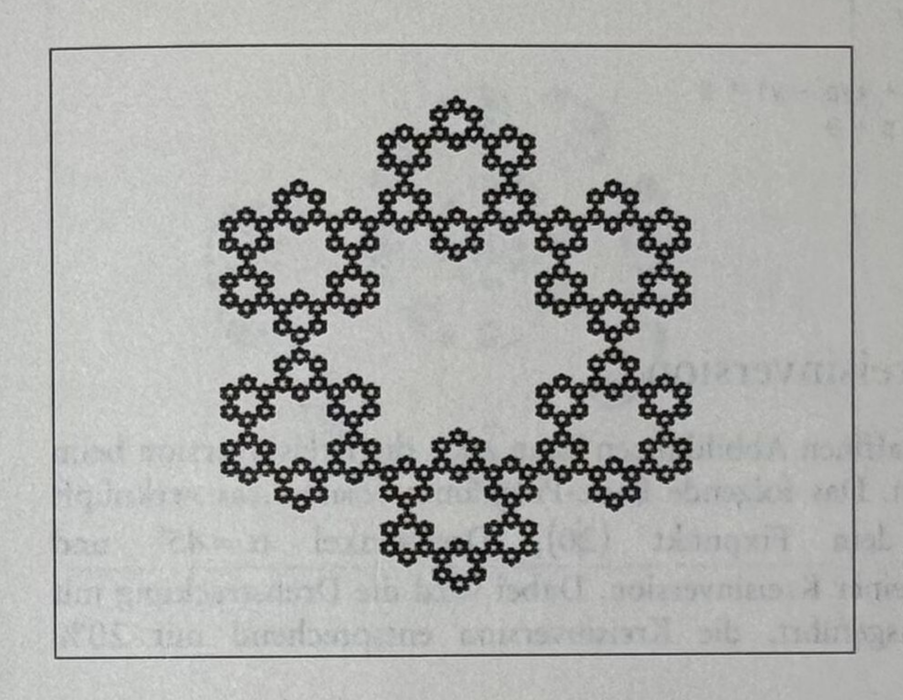


Bild 10.5 Drehsymmetrisches Sechseck mit eingeschlossener Schneeflockenkurve

Das Chaosspiel funktioniert hier wieder wie beim Abschnitt 10.1. Jedoch wird von der jeweiligen Verbindungsstrecke nicht der Mittelpunkt, sondern der Punkt mit der Entfernung 66.7% gewählt. Dies entspricht dem Teilungspunkt im Verhältnis 2:1. Zeitler [40] gibt die fraktale Dimension des Sechsecks mit

$$D = \frac{\ln 6}{\ln 3} = 1.6309$$

an. Das Vorgehen wird durch das Quick Basic-Programm `crystal6.bas` realisiert. Die Koordinaten des Sechsecks sind wieder für VGA-Auflösung vorgegeben.

```
' crystal6.bas

CONST N = .667
DIM p AS INTEGER
DIM i AS LONG
DIM x, y, xp, yp AS SINGLE

SCREEN 12: CLS
```

```
LINE (0, 0)-(639, 479), , B

RANDOMIZE TIMER
x = 0: y = 0
FOR i = 1 TO 100000
    p = INT(RND * 6)
    SELECT CASE p
      CASE 0: xp = 320: yp = 40
      CASE 1: xp = 508: yp = 137
      CASE 2: xp = 508: yp = 334
      CASE 3: xp = 320: yp = 430
      CASE 4: xp = 132: yp = 334
      CASE 5: xp = 132: yp = 137
    END SELECT
    x = x + (xp - x) * N: y = y + (yp - y) * N
    IF i > 10 THEN PSET (x, y), p + 9
NEXT i
a$ = INPUT$(1): SCREEN 0
END
```

10.5 Beispiel einer Kreisinversion

Neben den bisher genannten affinen Abbildungen kann auch die Kreisinversion beim Chaosspiel verwendet werden. Das folgende Basic-Programm schnecke.bas verknüpft eine Drehstreckung mit dem Fixpunkt $(2|0)$, Drehwinkel $\alpha = 45^\circ$ und Streckungsfaktor $c = 0.9$ mit einer Kreisinversion. Dabei wird die Drehstreckung mit 80% Wahrscheinlichkeit ausgeführt, die Kreisinversion entsprechend mit 20% Wahrscheinlichkeit.

```
'schnecke.bas
'Chaosspiel mit Kreisinversion

CONST PI = 3.14152653#
CONST c = .9: 'Kontraktionsfaktor
CONST alpha = PI/4: 'Drehwinkel
DIM a, b, x, x1, y AS SINGLE
DIM i AS INTEGER

SCREEN 12: CLS
WINDOW (-.6, -1.8)-(4.2, 1.6)
RANDOMIZE TIMER

a = c * COS(alpha): b = c * SIN(alpha)
x = 2: y = 0
FOR i = 1 TO 32000
  IF RND < .8 THEN
    x1 = x
    x = a * x - b * y + 2 - 2 * a
    y = b * x1 + a * y - 2 * b
```

```
  ELSE
    u = x * x + y * y
    x = x / u: y = y / u
  END IF
  PSET (x, y), 1 + u
NEXT i
a$ = INPUT$(1): SCREEN 0
END
```

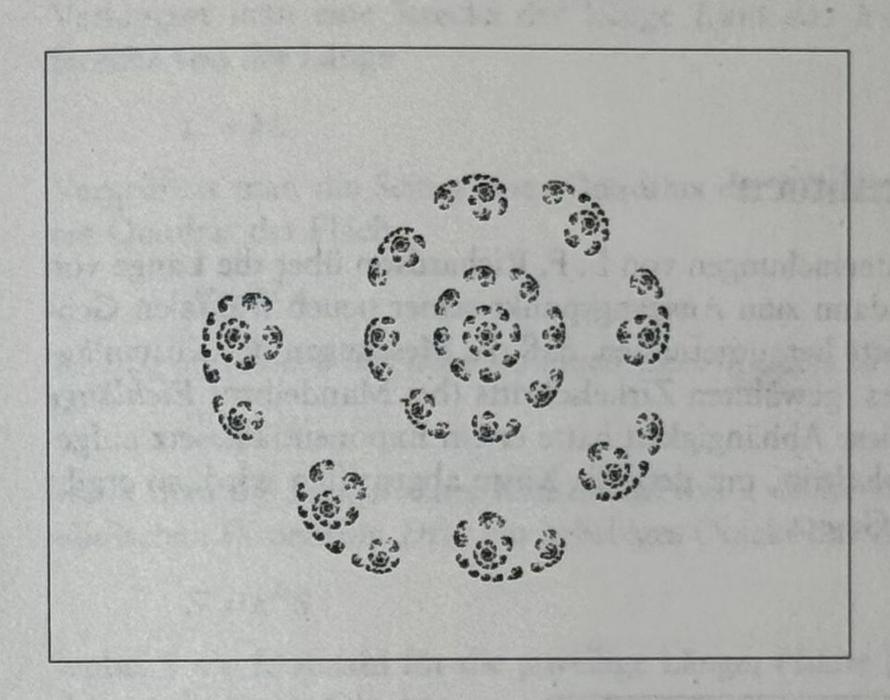
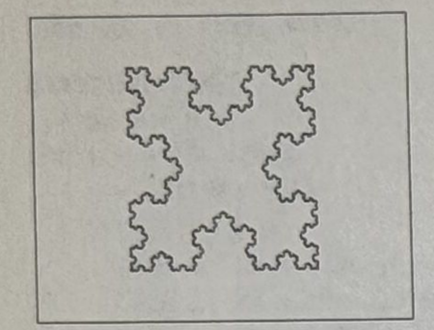


Bild 10.6 Schnecke mittels Chaosspiel



11 Fraktale Kurven

11.1 Länge von Küstenlinien

1967 fand Mandelbrot die Untersuchungen von L. F. Richardson über die Länge von Grenz- und Küstenlinien, die dann zum Ausgangspunkt seiner neuen fraktalen Geometrie wurden. Richardson hatte herausgefunden, daß die Messungen der Küstenlängen deutlich von der Größe des gewählten Zirkelschritts (bei Mandelbrot *Eichlänge* genannt) abhängig war. Für diese Abhängigkeit hatte er ein Exponentialgesetz aufgestellt. Ist s der gewählte Zirkelschritt, mit dem die Küste abgegriffen wird, so ergibt sich die Länge $L(s)$ nach dem Gesetz

$$L(s) = as^{1-D}$$

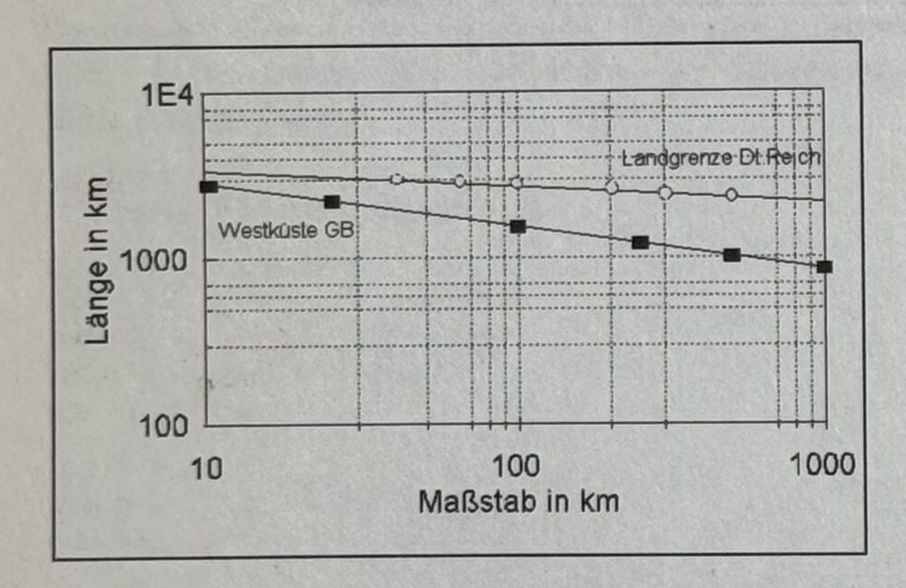


Bild 11.1 Küstenlängen nach Lewis Richardson

Das Bild 11.1 zeigt die Messungen für die englische Küste und die Grenze des deutschen Reichs von 1910 in einem doppelt-logarithmischen Koordinatensystem. Daraus läßt sich über die Steigung die fraktale Dimension zu $D = 1.26$ bzw. $D = 1.16$ berech-

nen. Der Wert $D = 1.26$ für die englische Westküste korrespondiert nahezu mit dem Wert $D = \frac{\ln 4}{\ln 3} = 1.262$ für die Schneeflockenkurve, wie in Abschnitt 11.8 gezeigt wird.

11.2 Die fraktale Dimension

Verlängert man eine Strecke der Länge L auf das k-fache, so ist die resultierende Strecke von der Länge

$$L' = kL$$

Vergrößert man die Seiten eines Quadrats der Fläche A auf das k-fache, so entsteht ein Quadrat der Fläche

$$A' = k^2 A$$

Analog ergibt sich das neue Volumen eines Würfels zu

$$V' = k^3 V$$

wenn man die Länge seiner Kanten auf das k-fache streckt. Somit folgt in jeder Euklidischen Dimension D für ein beliebiges Objekt das Gesetz

$$S' = k^D S$$

wobei S die Maßzahl für die jeweilige Länge, Fläche bzw. das Volumen ist. Gilt für das resultierende Objekt

$$S' = NS$$

so folgt durch Einsetzen und Logarithmieren

$$NS = k^D S \quad \Rightarrow \quad D = \frac{\ln N}{\ln k}$$

Diese Formel, die für die Euklidschen Dimensionen 1 bis 3 erklärt wurde, soll nun auch auf beliebige fraktale Objekte angewandt werden. Da das Verhältnis zweier Logarithmen i.a. nicht ganzzahlig ist, erhält man nach der neuen Definition reelle Maßzahlen für die Dimension. Der so definierte Dimensionsbegriff wird nach Vorschlag von Mandelbrot fraktal genannt.

Ein Objekt heißt *selbstähnlich*, wenn es bei der Teilung der Kanten in r gleiche Abschnitte in N gleiche Teile zerfällt. In diesem Fall gilt für den Streckfaktor $k = \frac{1}{r}$ und die fraktale Dimension läßt sich schreiben als

$$D = \frac{\ln N}{\ln \frac{1}{r}}$$

Umformen liefert $D \ln\frac{1}{r} = \ln\left(\frac{1}{r}\right)^D = \ln N$ oder nach dem Delogarithmieren $Nr^D = 1$. Ist das Objekt nicht selbstähnlich, so kann diese Form nicht direkt angewandt werden. Man muß dann den Grenzwert

$$D = \lim_{r \to 0} \frac{\ln N(r)}{\ln \frac{1}{r}}$$

bilden. Dieses Maß entspricht der Box-Counting-Dimension, die ein Spezialfall der Hausdorff-Besikowitch-Dimension ist.

11.3 Die Cantor-Menge

Als einfaches Beispiel wird die Cantor-Menge behandelt. Dies ist eine Menge von reellen Zahlen, benannt nach dem deutschen Mathematiker und Begründer der Mengenlehre, Georg Cantor. Sie entsteht durch einen Grenzprozeß aus der Einheitsstrecke [0;1]. Zuerst entfernt man das mittlere Drittel des Intervalls, dann vom ersten und dritten Drittel wieder das mittlere Drittel und so fort. Dieses Verfahren wird unendlich oft fortgesetzt. Die verbleibenden Zahlen der Einheitsstrecke stellen die Cantor-Menge dar.

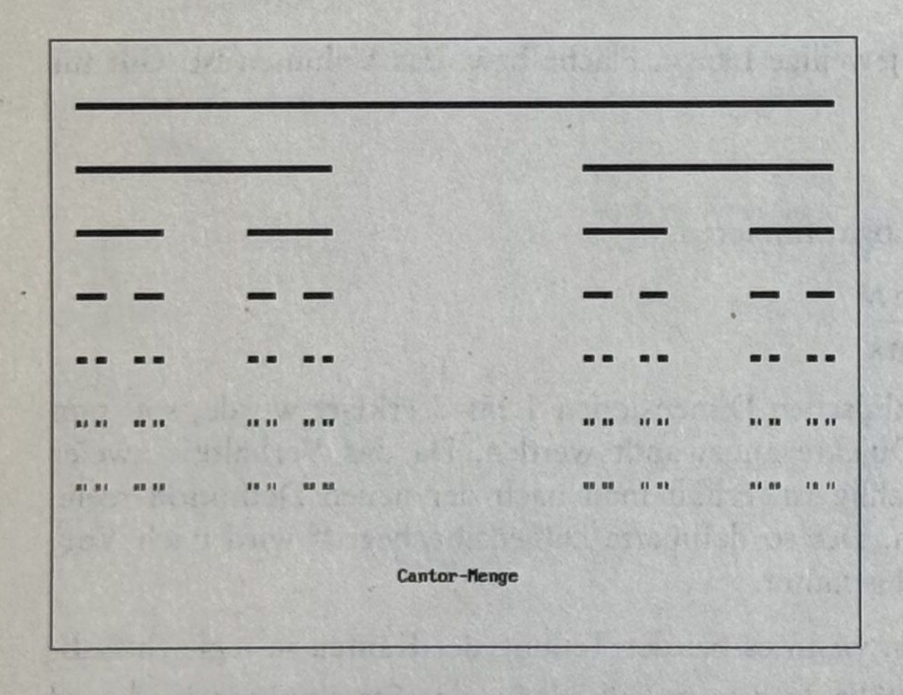


Bild 11.2 Cantormenge

Das Bild 11.2 kann mit dem Quick Basic-Programm `cantor.bas` erstellt worden. Wegen der beschränkten Punktezahl des Bildschirms können nur die ersten Schritte des Cantorschen Drittelungsverfahren veranschaulicht werden.

```
'cantor.bas
 "Cantor-Menge

CONST dx = 1 / 640
DIM i, j, p, q AS INTEGER
DIM x, x1, y AS SINGLE

SCREEN 12: CLS
LINE (0, 0)-(639, 479), , B

FOR i = 0 TO 4
   LINE (20, 50 + i)-(620, 50 + i), 1
NEXT i
FOR j = 0 TO 639
   x = j * dx: x1 = x
   FOR i = 1 TO 6
   y = 3 * (.5 - ABS(x - .5))
   x = y
   IF (x >= 0) AND (x <= 1) THEN
     p = x1 * 600 + 20: q = 50 * i + 50
     FOR k = 0 TO 4
       PSET (p, q + k), 1
     NEXT k
   END IF
   NEXT i
NEXT j
LOCATE 27, 35: PRINT "Cantor-Menge"
e$ = INPUT$(1): SCREEN 0
END
```

Bei einer Drittelung, d.h. bei einem Streckfaktor $k=\frac{1}{3}$ zerfällt die Cantor-Menge in $N=2$ selbstähnliche Teile. Die fraktale Dimension ist daher

$$D=\frac{\ln 2}{\ln 3}=0.6309$$

Die euklidische Dimension ist Null, da sie nur Punkte enthält und daher auch Cantor-Staub genannt wird.

11.4 Die Teufelstreppe

Die Teufelstreppe (englisch *devil's staircase*) steht in enger Beziehung zur Cantor-Menge. Sie ist eine monoton steigende Kurve, definiert auf dem Einheitsintervall, die überall dort, wo beim Cantor-Verfahren ein Intervall-Drittel entfernt wird, ein waagrechtes Plateau aufweist. Die Höhe der Treppe ist ebenfalls eins. Wie man sieht, ist die Treppenkurve nicht selbstähnlich; es läßt sich aber zeigen, daß die fraktale Dimension gleich 1 ist (siehe [20]). Da die fraktale Dimension nicht die euklidische übersteigt, ist die Kurve nicht im Sinne von Mandelbrot fraktal.

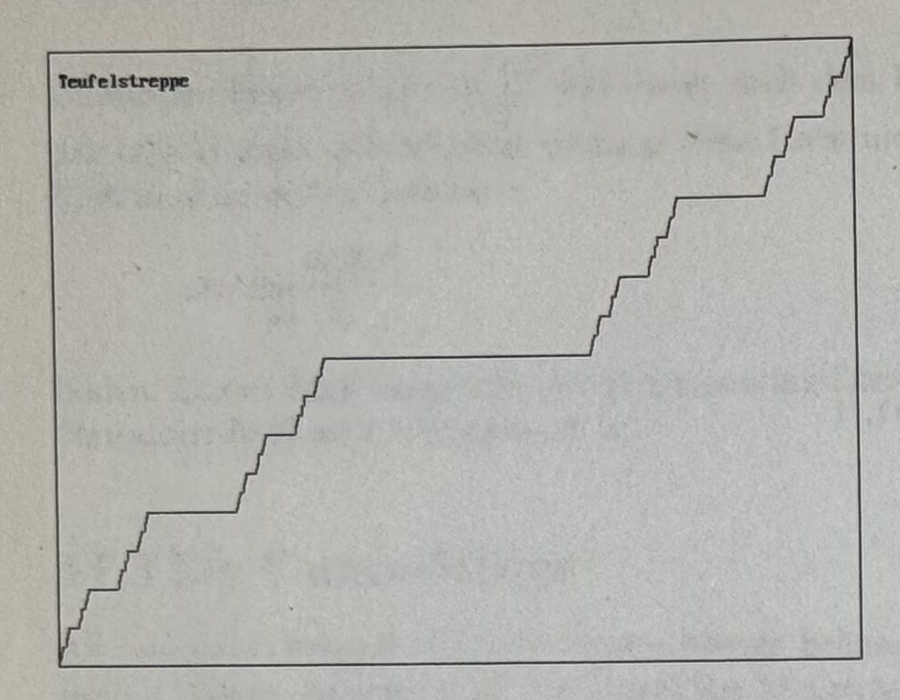


Bild 11.3 Teufelstreppe

Die Treppe wird vom Quick Basic-Programm `devil.bas` gezeichnet.

```
'devil.bas
 "Teufelstreppe

DIM i AS INTEGER
DIM xl(10), xr(10), yr(10), yl(10)

SCREEN 12: CLS
LINE (0, 0)-(639, 479), , B

i = 8
xl(i) = 0: xr(i) = 639
yl(i) = 479: yr(i) = 0
GOSUB recursion
LOCATE 2, 2: PRINT "Teufelstreppe"
e$ = INPUT$(1): SCREEN 0
END
recursion:
IF i = 1 THEN
   LINE (xl(1), yl(1))-(xr(1), yr(1))
ELSE
   i = i - 1
   xl(i) = xl(i + 1)
   yl(i) = yl(i + 1)
   xr(i) = (2 / 3 * xr(i + 1) + 4 / 3 * xl(i + 1)) / 2
   yr(i) = (yr(i + 1) + yl(i + 1)) / 2
   GOSUB recursion
   xl(i) = (4 / 3 * xr(i + 1) + 2 / 3 * xl(i + 1)) / 2
   yl(i) = (yr(i + 1) + yl(i + 1)) / 2
   LINE (xl(i), yl(i))-(xr(i), yr(i))
   xr(i) = xr(i + 1)
```

```
  yr(i) = yr(i + 1)
  GOSUB recursion
  i = i + 1
END IF
RETURN
```

11.5 Cantor-Quadrat

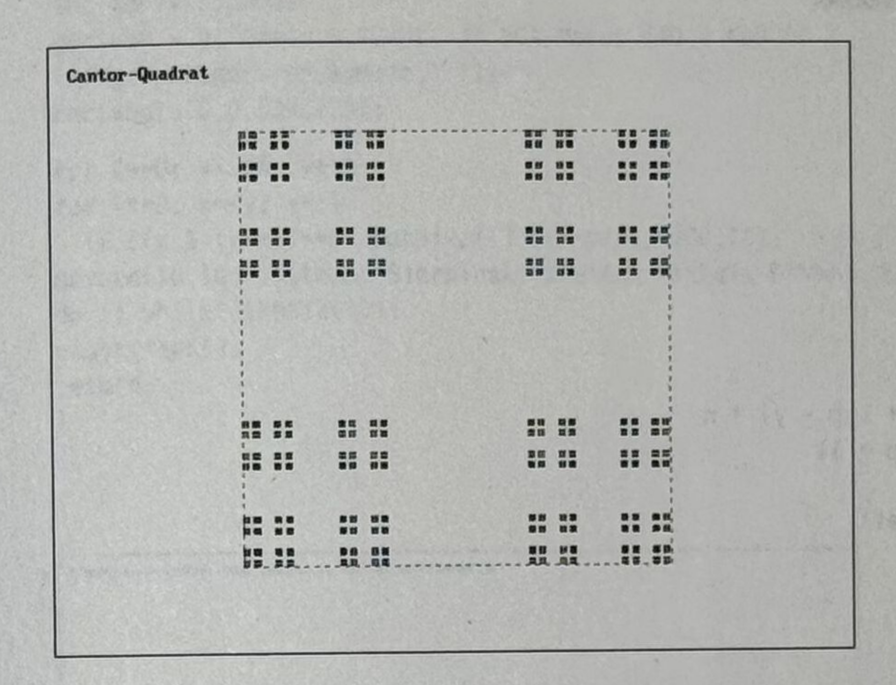


Bild 11.4 Cantor-Quadrat

Das Cantor-Quadrat ist das zweidimensionale Analogon zur Cantormenge; d.h. das Cantorverfahren wird auf Länge und Breite eines Quadrates angewandt. Bei der Drittelung der Seiten entstehen so je 4 selbstähnliche Quadrate; daraus leitet sich die fraktale Dimension

$$D = \frac{\ln 4}{\ln 3} = 1.2619$$

her. Die Dimension kann auch direkt aus der Produktformel berechnet werden. Da das Cantor-Quadrat das Produkt $C \times C$ der Cantor-Menge C ist, ergibt sich direkt

$$D = 2\dim(C) = 2\frac{\ln 2}{\ln 3} = \frac{\ln 4}{\ln 3}$$

Das Cantor-Quadrat kann mit Hilfe des Chaosspiels erzeugt werden; eine Implementierung liefert das Quick Basic-Programm `cantquad.bas`.

```
'cantquad.bas
 "Cantor-Quadrat

CONST n = 1.33333
DIM p, xp, yp AS INTEGER
DIM i AS LONG
DIM x, y AS SINGLE

SCREEN 12: CLS
LINE (0, 0)-(639, 479), , B
LINE (150, 70)-(492, 411), , B, &HAAAA

RANDOMIZE TIMER
x = 0: y = 0
FOR i = 1 TO 40000!
    p = INT(RND * 4)
    SELECT CASE p
    CASE 0: xp = 405: yp = 325
    CASE 1: xp = 235: yp = 325
    CASE 2: xp = 235: yp = 155
    CASE 3: xp = 405: yp = 155
    END SELECT
    x = x + (xp - x) * n: y = y + (yp - y) * n
    IF i > 20 THEN PSET (x, y), p + 11
NEXT i
LOCATE 2, 3: PRINT "Cantor-Quadrat"
A$ = INPUT$(1): SCREEN 0
END
```

11.6 Sierpinski-Dreieck

Das Sierpinski-Dreieck entsteht aus einem gleichseitigen Dreieck, aus dem das Mittendreieck entfernt wird. Dadurch zerfällt das Dreieck in 3 weitere Teildreiecke, aus denen wiederum die Mittendreiecke entfernt werden. Der Grenzwert des Verfahrens liefert das gesuchte Dreieck.

Daraus folgt die Selbstähnlichkeit der Figur. Nimmt man ein beliebiges Teildreick und streckt eine Seite mit dem Streckfaktor $k=2$, so erhält die entstehende Figur $N=3$ gleiche Teildreiecke. Damit ergibt sich die frakale Dimension zu

$$D=\frac{\ln 3}{\ln 2}=1.5850$$

Damit liegt die Dimension des Sierpinski-Dreiecks etwa in der Mitte zwischen einer Geraden und einer Fläche. Das folgende Turbo C-Programm `sierp2.c` realisiert das Sierpinski-Dreieck mit Hilfe von Binär-Adressen.

```
/* sierp2.c */
/* Sierpinski-Dreieck mittels Binär-Arithmetik */

#include <stdio.h>
#include <graphics.h>
#include <conio.h>

void main(void)
{
int x,y;
int gdriver,gmode;
gdriver = 9; gmode = VGAHI; /* VGA-Modus 640 x 480 */
initgraph(&gdriver,&gmode," ");
rectangle(0,0,639,479);

for (y=0; y<256; y++)
for (x=0; x<=y; x++)
  if ((x & (y-x))==0) putpixel(320+x-y/2,y+100,14);
moveto(10,10);outtext("Sierpinski-Dreieck mittels Binärarithmetik");
do {} while(!(kbhit()));
closegraph();
return;
}
```

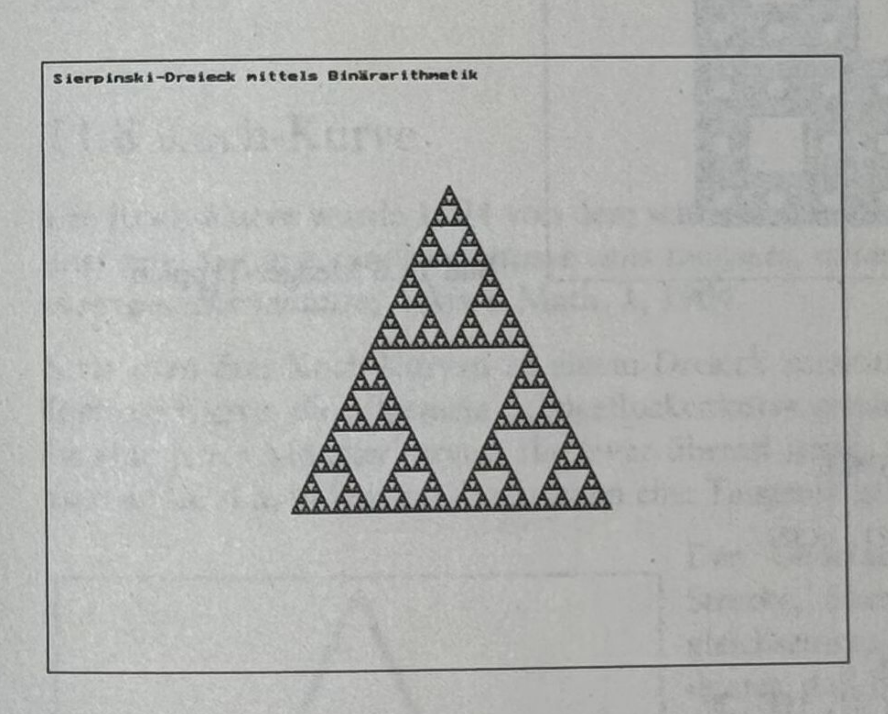


Bild 11.5 Sierpinski-Dreieck

11.7 Menger-Teppich

Der Menger-Teppich, auch Sierpinski-Teppich genannt, wird analog zum Sierpinski-Dreieck konstruiert. Die Drittelung der Seiten liefert 9 Teilquadrate, von denen jeweils das mittlere entfernt wird. Auf die verbleibenden 8 Quadrate wird Vorgang

wiederholt. Setzt man das Teilungsverfahren unbeschränkt wird, so ergibt sich der Menger-Teppich als Grenzwert der verbleibenden Punkte.

Da eine Drittelung $r = \frac{1}{3}$ insgesamt $N = 8$ Teile liefert, berechnet sich die fraktale Dimension zu

$$D = \frac{\ln 8}{\ln 3} = 1.8928$$

Das Quick Basic-Programm `menger2.bas` arbeitet ähnlich wie das Chaosspiel, die Teilquadrate werden aber nicht ausgelost, sondern systematisch durch Rekursion erfaßt.

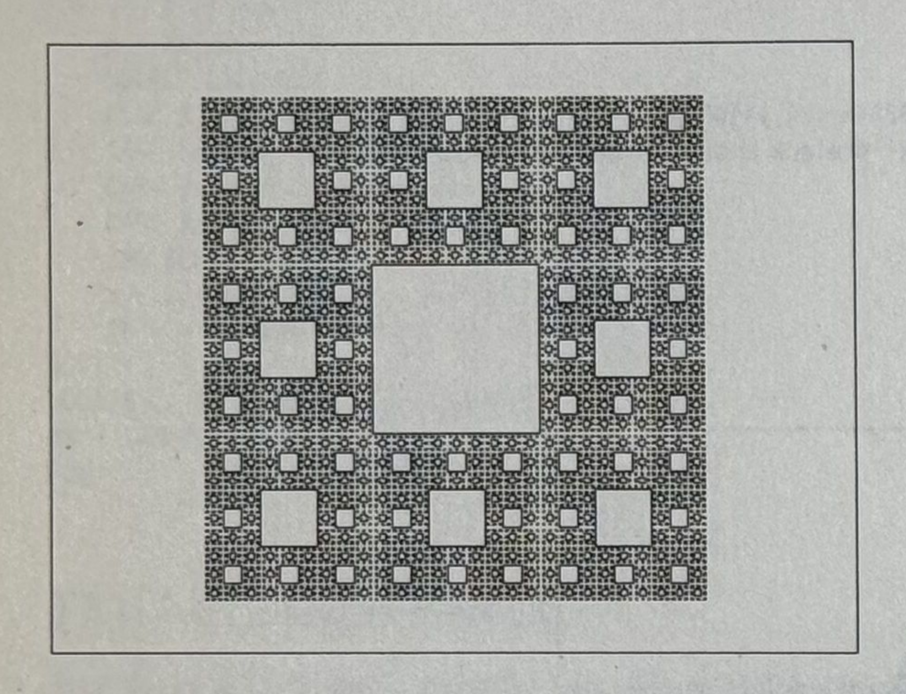


Bild 11.6 Menger-Teppich

```
"menger2.bas

CONST p = 4: 'Rekursionstiefe
DIM h, x, x1, x2, y, y1, y2 AS SINGLE
DIM j, g, m AS INTEGER
DIM x(8), y(8), z(8), a(32), b(32), c(32)

SCREEN 12: CLS
WINDOW (-1.6, -1.2)-(1.6, 1.2)
FOR j = 1 TO 4: READ f(j): NEXT j
LINE (-1 / 3, 1 / 3)-(1 / 3, -1 / 3), f(1), BF

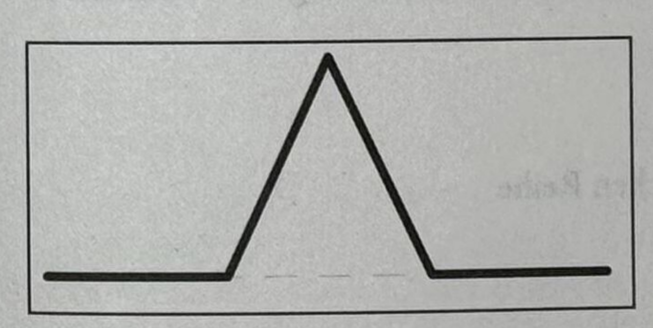
x = 0: y = 0: m = 0: g = 1
DO WHILE m >= 0
  IF g < p + 1 THEN
    x(1) = (x + 2) / 3: y(1) = (y + 2) / 3
    x(2) = (x + 2) / 3: y(2) = (y - 2) / 3
    x(3) = (x - 2) / 3: y(3) = (y + 2) / 3
```

```
    x(4) = (x - 2) / 3: y(4) = (y - 2) / 3
    x(5) = (x + 2) / 3: y(5) = y / 3
    x(6) = (x - 2) / 3: y(6) = y / 3
    x(7) = x / 3: y(7) = (y + 2) / 3
    x(8) = x / 3: y(8) = (y - 2) / 3
    h = 3 ^ (-g - 1)
    FOR j = 1 TO 8
      x1 = x(j) - h: y1 = y(j) + h
      x2 = x(j) + h: y2 = y(j) - h
      LINE (x1, y1)-(x2, y2), f(g), BF
    NEXT j
    x = x(1): y = y(1): g = g + 1
    FOR j = 2 TO 8
      m = m + 1
      a(m) = x(j): b(m) = y(j): c(m) = g
    NEXT j
   ELSE
     x = a(m): y = b(m): g = c(m)
     m = m - 1
   END IF
   LOOP
a$ = INPUT$(1): SCREEN 0
END
DATA 1,3,11,9
```

11.8 Koch-Kurve

Die Koch-Kurve wurde 1904 von dem schwedischen Mathematiker Helge von Koch angeben: *Sur une courbe continue sans tangente, obtenue par une construction géometrique élémentaire*, Arkiv f. Math. 1, 1904.

Setzt man drei Koch-Kurven zu einem Dreieck zusammen, so erhält man eine sternförmige Kurve, die allgemein Schneeflockenkurve genannt wird. Die Kochsche Kurve ist eine jener Monsterkurven, die zwar überall stetig, aber an keiner Stelle differenzierbar ist; d.h. in keinem Punkt kann eine Tangente gelegt werden.



Der Generator der Kochkurve ist eine Strecke, über deren mittlerem Drittel ein gleichseitiges Dreieck gesetzt ist. Dies bedeutet, daß bei einer Drittelung der Strecke 4 Teilabschnitte erzeugt werden. Die fraktale Dimension ist daher $D = \frac{\ln 4}{\ln 3} = 1.2619$

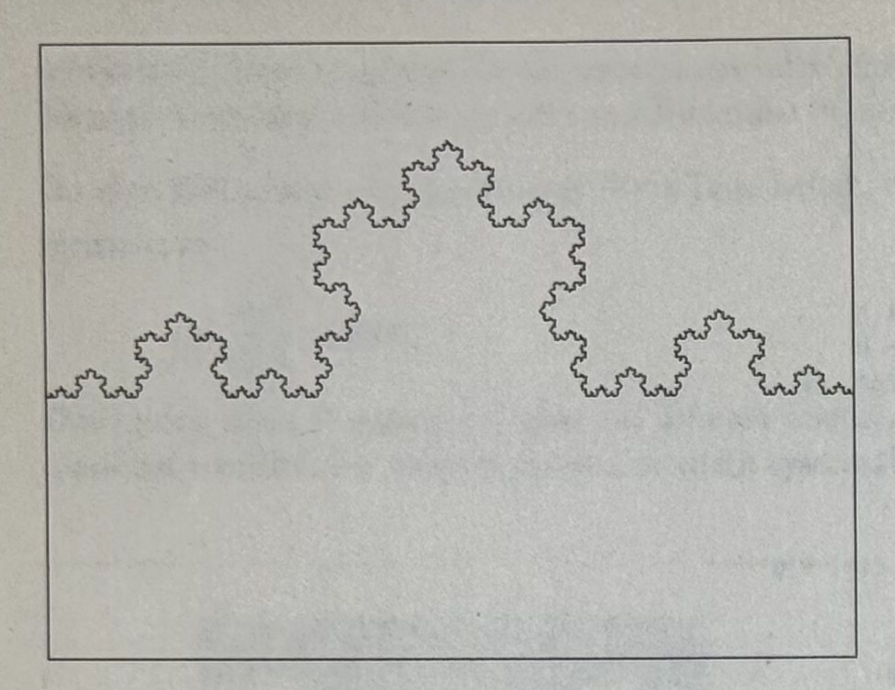


Bild 11.7 Schneeflockenkurve

Die Länge der Schneeflocke wächst über jede endliche Schranke; d.h. der Umfang ist unendlich. Dies sieht man an einer kleinen Rechnung. Hat das Ausgangsdreieck die Länge 1, so ist je Seite $s_0 = \frac{1}{3}$. Nach der n-ten Drittelung gilt dann für die Länge $s_n = \left(\frac{1}{3}\right)^n$. Da sich die Anzahl der Seiten bei jedem Schritt vervierfacht, erhöht sich ihre Zahl nach dem Gesetz $a_n = 3 \cdot 4^n$. Der Umfang nach der n-ten Teilung ist damit

$$U_n = a_n s_n = 3\left(\frac{4}{3}\right)^n$$

Daraus folgt,daß der Umfang der Grenzkurve nicht beschränkt ist

$$U = \lim_{n \to \infty} U_n \to \infty$$

Die Fläche der Schneeflockenkurve wird, wie folgt, bestimmt. Die Fläche des Ausgangsdreiecks ist wegen der Gleichseitigkeit $A_0 = \frac{1}{4}\sqrt{3}$. Bei jedem Drittelungsschritt addieren sich die a_n Flächen der verkleinerten Dreiecke, dies zeigt die Rekursionsformel

$$A_n = A_{n-1} + a_n \cdot \frac{1}{4}\sqrt{3} \cdot s_n^2$$

Dies ist die Rekursionsformel der geometrischen Reihe

$$A_n = A_0 + \frac{1}{12}\sqrt{3} \sum_{i=0} \left(\frac{4}{9}\right)^i$$

mit dem Ergebnis $A_n = \frac{2}{5}\sqrt{3} = \frac{8}{5} A_0$

Damit ist der Flächeninhalt der Grenzkurve das 1.6-fache der Ausgangsfigur! Obwohl die Länge der Kurve beliebig groß wird, bleibt der eingeschlossene Inhalt endlich. Die Erklärung dafür liefert die fraktale Dimension $1<D<2$.

Ein Basic-Programm zum Zeichnen der Kochkurve ist `koch.bas`.

```
'koch.bas

CONST PI = 3.141593
CONST p = 5:  'Rekursionsstufe
DIM m, n, k, l, s AS INTEGER
DIM h, x, y AS SINGLE
DIM t(p)

SCREEN 12: CLS
WINDOW (0, -.3)-(1, .4)
LINE (0, -.3)-(1, .4), , B

h = 3 ^ (-p)
PSET (0, 0)
FOR n = 0 TO 4 ^ p - 1
   m = n
   FOR l = 0 TO p - 1
     t(l) = m MOD 4: m = m \ 4:
   NEXT l
   s = 0
   FOR k = 0 TO p - 1
     s = s + (t(k) + 1) MOD 3 - 1
   NEXT k
   x = x + COS(PI * s / 3) * h
   y = y + SIN(PI * s / 3) * h
   LINE -(x, y), 1 + n MOD 5
NEXT n
e$ = INPUT$(1): SCREEN 0
END
```

Bringt man die Koch- Kurve an den Seiten eines Quadrats innen an, so erhält man das Bild 11.8.

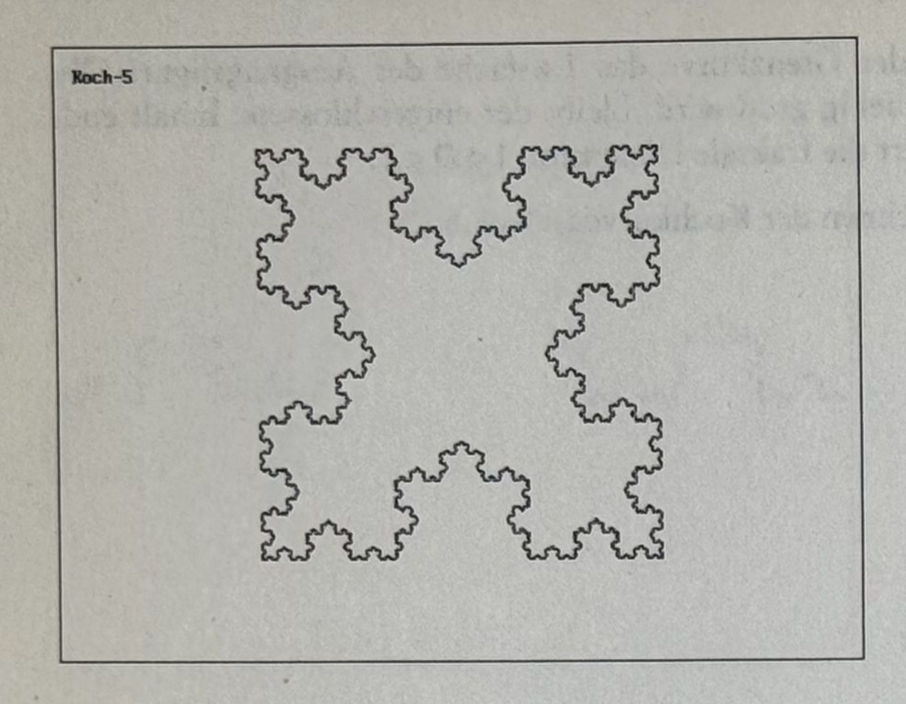


Bild 11.8 Kochkurve im Quadrat

11.9 Koch-2

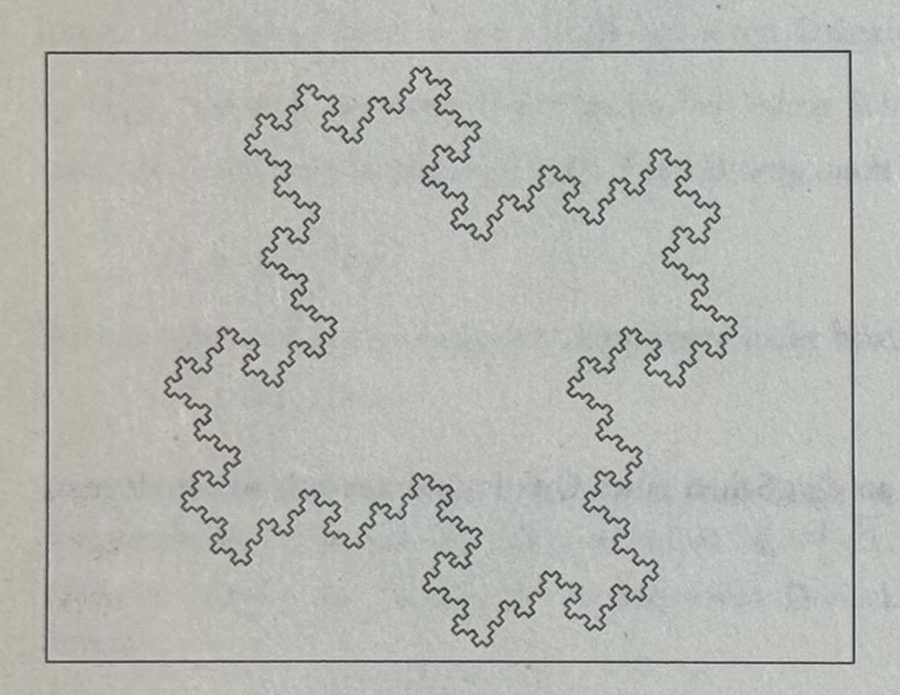
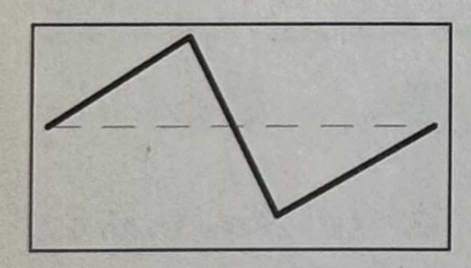


Bild 11.9 Koch-2



Die Koch-Kurve kann auf vielfältigste Art verallgemeinert werden. Durch den Generator hält man die kochähnliche Kurve Koch-2, sie wird von Mandelbrot *Alternative Koch-Kurve* genannt (siehe Bild 11.9). Geht man von der Einheitsstrecke aus, so sind die Eckpunkte des Generators (0.4,0.2) und (0.6,-0.2). Der Generator besteht damit aus

$N = 3$ Strecken der Länge $\frac{1}{r} = \sqrt{5}$. Die fraktale Dimension ist damit $D = \frac{\ln 3}{\ln \sqrt{5}} = 1.3652$

Für alle Koch-Kurven läßt sich ein universelles Basic-Programm nach Lauwerier [14] angeben, das nach Eingabe der Anfangskurve, *Initiator* genannt und des Generators die zugehörige Kurve zeichnet.

```
'koch2.bas

CONST p = 5: 'Rekursionstiefe
CONST u = 4: v = 3
DIM a1, b1, i, j, k, l, m, m1, m2, n AS INTEGER
DIM x1, y1 AS SINGLE
DIM a, b(0 TO u) AS SINGLE
DIM c, d(0 TO v) AS SINGLE
DIM x(4096), y(4096)

SCREEN 12: CLS
WINDOW (-2.5, -2)-(2.5, 2)

FOR i = 0 TO u
   READ a(i), b(i)
NEXT i
FOR i = 1 TO v - 1
   READ c(i), d(i)
NEXT i
c(0) = 0: d(0) = 0: x(0) = 0: y(0) = 0
x(v ^ p) = 1: y(v ^ p) = 0
FOR i = 0 TO p - 1
   FOR j = 0 TO v ^ p - 1 STEP v ^ (p - i)
      m1 = j + v ^ (p - i)
      x1 = x(m1) - x(j)
      y1 = y(m1) - y(j)
      FOR k = 1 TO v - 1
         m2 = j + k * v ^ (p - i - 1)
         x(m2) = x1 * c(k) - y1 * d(k) + x(j)
         y(m2) = y1 * c(k) + x1 * d(k) + y(j)
      NEXT k
   NEXT j
NEXT i
PSET (a(0), b(0))
FOR m = 0 TO u - 1
   a1 = a(m + 1) - a(m)
   b1 = b(m + 1) - b(m)
   FOR n = 0 TO v ^ p
      x = a1 * x(n) - b1 * y(n) + a(m)
      y = b1 * x(n) + a1 * y(n) + b(m)
      LINE -(x, y), 1 + m MOD 8
   NEXT n
NEXT m
```

```
e$ = INPUT$(1): END
DATA 1,1,-1,1,-1,-1,1,-1,1,1
DATA .4,.2,.6,-.2
```

Die kartesischen Koordinaten des Initiators sind in der vorletzten DATA-Zeile,die Punkte des Generators in der letzten Zeile einzugeben. Zu beachten ist, daß der Kurvenzug des Initiators geschlossen sein muß, d.h. der letzte Punkt muß mit dem ersten übereinstimmen.

11.10 Koch 3

Als weiteres Beispiel für eine Koch-Kurve, soll die von dem Generator mit den Teilungspunkten (0.25,0.25) und (0.75,-0.25) erzeugte Kurve behandelt werden. Berechnet man nämlich die 3 Teilstrecken des Generators, so erkennt man, daß die Kurve nicht exakt selbstähnlich ist. Der Generator besteht aus 2 Strecken der Länge $\frac{1}{2\sqrt{2}}$ und einer Strecke der Länge $\frac{1}{\sqrt{2}}$. Es tritt hier das Problem auf, daß die fraktale Dimensionsformel

$$Nr^D = 1$$

verallgemeinert werden muß. Treten außer den N Strecken der Länge $\frac{1}{r}$ auch noch M Strecken der Länge $\frac{1}{s}$ auf, so gilt die verallgemeinerte Dimensionsformel

$$Nr^D + Ms^D = 1$$

Diese Formel ist nichtlinear in D und daher meist nur numerisch zu lösen.

Mit den oben angegebenen Werten ergibt sich

$$2\left(\frac{1}{2\sqrt{2}}\right)^D + \left(\frac{1}{\sqrt{2}}\right)^D = 1$$

Schreibt man die Wurzeln als Zweierpotenz, so ergibt sich

$$2\left(2^{-\frac{3}{2}}\right)^D + \left(2^{-\frac{1}{2}}\right)^D = 1$$

Setzt man $x = \left(2^{\frac{1}{2}}\right)^D = 2^{\frac{D}{2}}$, so ergibt sich

$$2 \cdot \frac{1}{x^3} + \frac{1}{x} = 1 \text{ oder } x^3 - x^2 - 2 = 0.$$

Die Lösung dieser Gleichung liefert $x = 1.6956$ und damit für die gesuchte Diemension

$$D = 2\,\mathrm{ld}x = 1.5236.$$

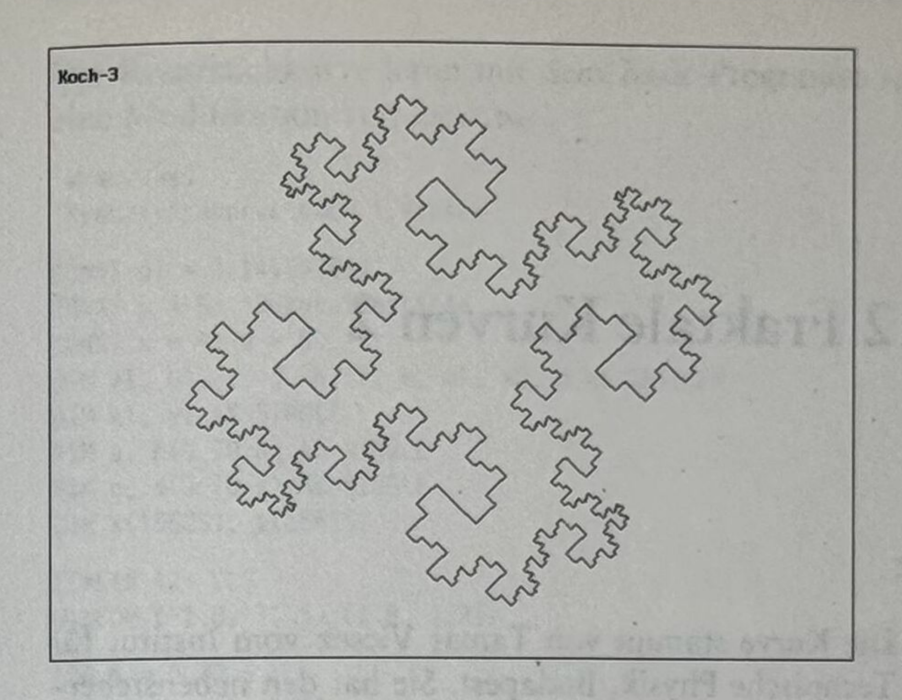
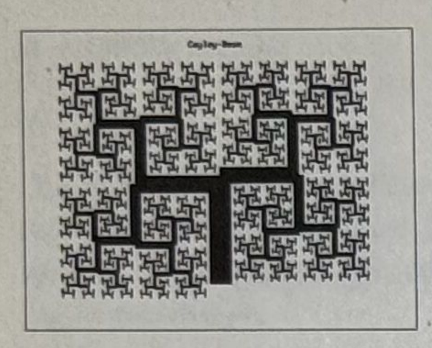
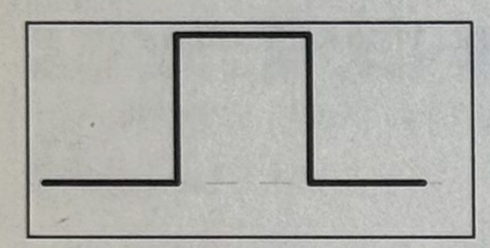


Bild 11.10 Koch-3 Kurve



12 Fraktale Kurven 2

12.1 Kreuzstichkurve



Die Kurve stammt von Tamas Vicsek vom Institut für Technische Physik, Budapest. Sie hat den nebenstehenden Generator und kann deshalb als Koch-ähnlich angesehen werden. Bei der Drittelung mit $\frac{1}{r} = 3$ ergeben sich hier $N = 5$ Teilstrecken.

Daraus läßt sich die fraktale Dimension

$$D = \frac{\ln 5}{\ln 3} = 1.4650$$

ableiten. Wirkt der Generator auf einem Quadrat nach außen, so erhält man Bild 12.2; Bild 12.1 verwendet die Einheitsstrecke als Initiator.

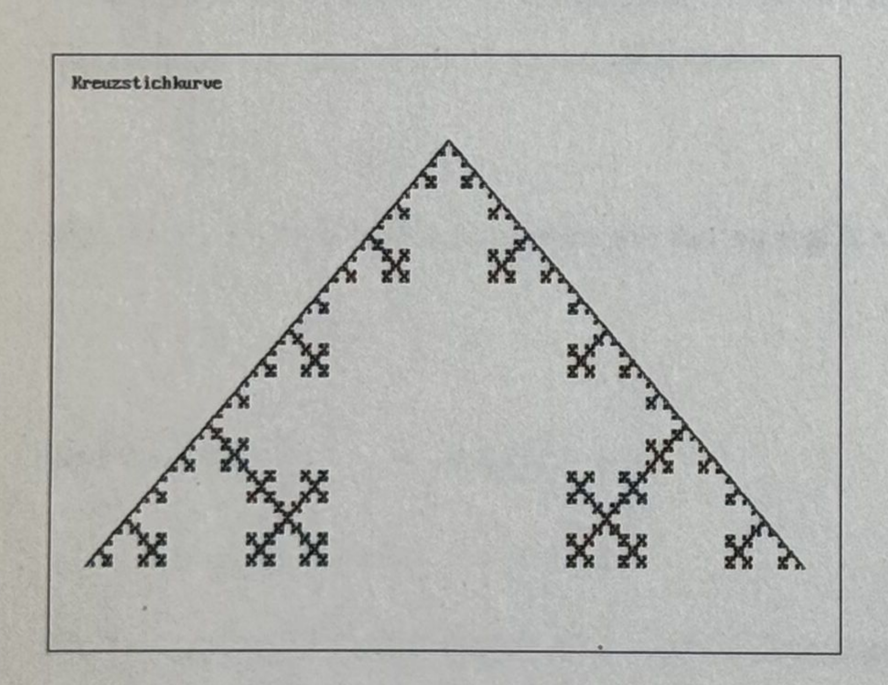


Bild 12.1 Kreuzstichkurve über Einheitsstrecke

Die Kreuzstichkurve kann mit dem Basic-Programm `kreuz.bas` erzeugt werden; es ist eine Modifikation von `koch2.bas`.

```
'kreuz.bas
'Kreuzstichkurve nach T.Vicsek

CONST pi = 3.14159265#
CONST p = 5: 'Rekursionstiefe
CONST u = 4: v = 5
DIM a1, b1, i, j, k, l, m, m1, m2, n AS INTEGER
DIM x1, y1 AS SINGLE
DIM a, b(0 TO u) AS SINGLE
DIM c, d(0 TO v) AS SINGLE
DIM x(15625), y(15625)

SCREEN 12: CLS
WINDOW (-1.8, -1.3)-(1.8, 1.3)

FOR i = 0 TO u
   READ a(i), b(i)
NEXT i
FOR i = 1 TO v - 1
   READ c(i), d(i)
NEXT i
c(0) = 0: d(0) = 0: x(0) = 0: y(0) = 0
x(v ^ p) = 1: y(v ^ p) = 0
FOR i = 0 TO p - 1
   FOR j = 0 TO v ^ p - 1 STEP v ^ (p - i)
      m1 = j + v ^ (p - i)
      x1 = x(m1) - x(j)
      y1 = y(m1) - y(j)
      FOR k = 1 TO v - 1
         m2 = j + k * v ^ (p - i - 1)
         x(m2) = x1 * c(k) - y1 * d(k) + x(j)
         y(m2) = y1 * c(k) + x1 * d(k) + y(j)
      NEXT k
   NEXT j
NEXT i
PSET (a(0), b(0))
FOR m = 0 TO u - 1
   a1 = a(m + 1) - a(m)
   b1 = b(m + 1) - b(m)
   FOR n = 0 TO v ^ p
      x = a1 * x(n) - b1 * y(n) + a(m)
      y = b1 * x(n) + a1 * y(n) + b(m)
      LINE -(x, y), 1 + m MOD 8
   NEXT n
NEXT m
e$ = INPUT$(1): END
DATA -1,-1, 1,-1, 1,1, -1,1, -1,-1
DATA .333,0, .333,.333, .667,.333, .667,0
```

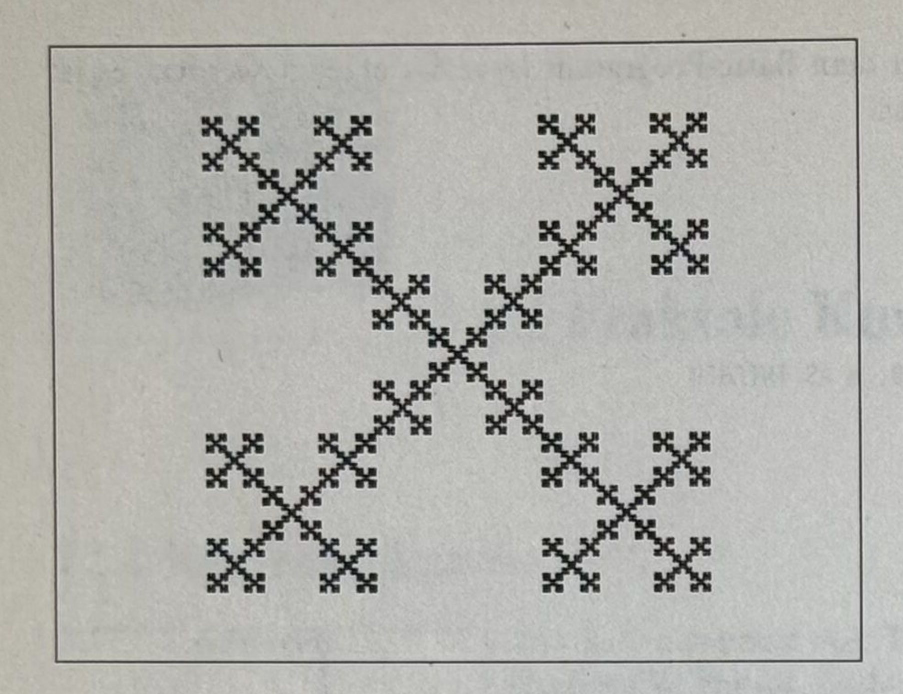


Bild 12.2 Kreuzstichkurve im Quadrat

12.2 Eiskurven

Der Initiator der Eiskurve besteht aus der Einheitsstrecke, der Generator erzeugt im Mittelpunkt der Strecke eine Lotstrecke der Länge $\frac{1}{3}$. Dies hat zur Folge, daß die Grenzkurve flächenfüllend ist. Dies folgt aus dem Grenzfall der Cesàro-Kurve für $\alpha = 90^\circ$ (vgl. Abschnitt 12.3).

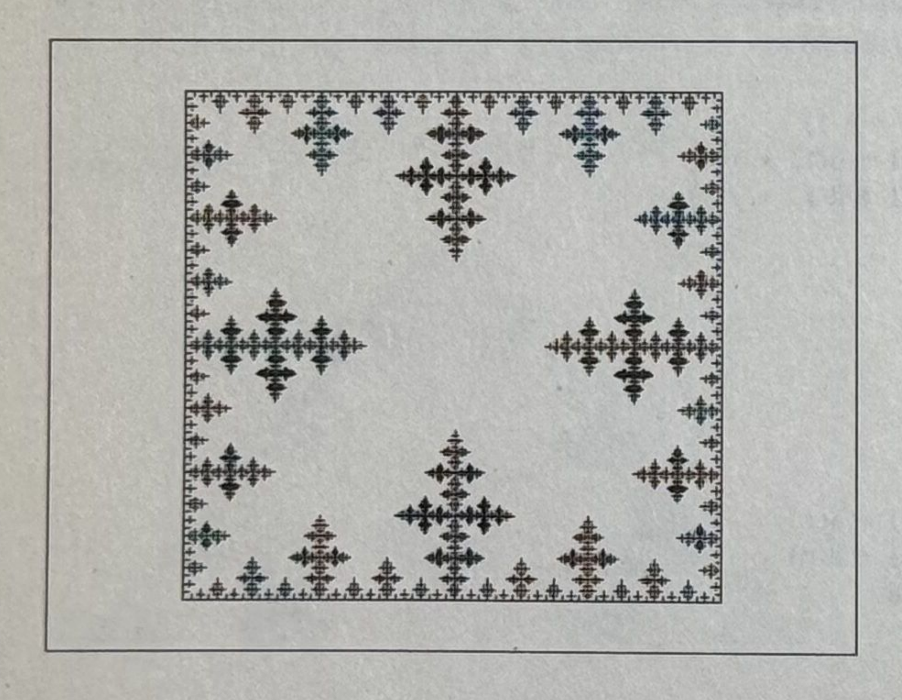


Bild 12.3 Eisquadrat

Als Initiator und Generator sind für das Eisquadrat folgende DATA-Zeilen in das Basic-Programm `kreuz.bas` oder `koch2.bas` einzusetzen:

```
DATA 1,1,-1,1,-1,-1,1,-1,1,1
DATA .5,0,.5,.333,.5,0
```

Für das folgende Eisdreieck ist einzusetzen

```
DATA 0.433,0,-0.433,0.5,-0.433,-0.5,0.433,0
DATA .5,0,.375,.2165,.5,0,.625,.2165,.5,0
```

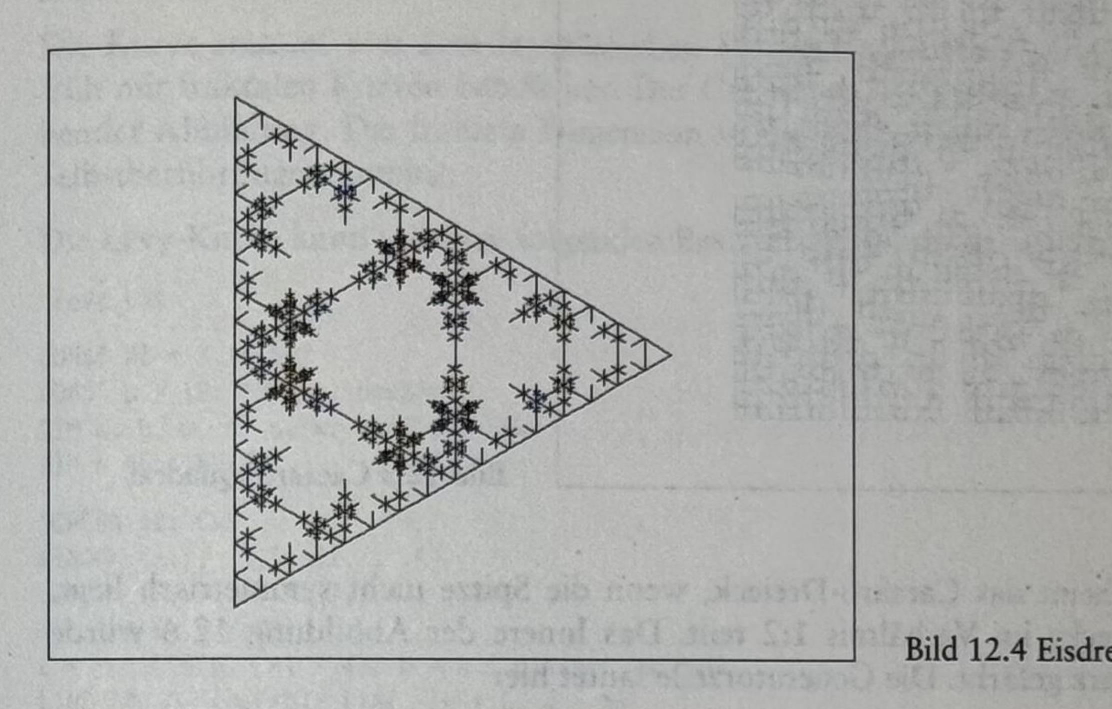
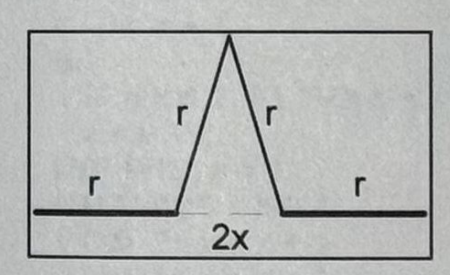


Bild 12.4 Eisdreieck

12.3 Caesàro-Kurve



Die Caesàro-Kurve hat den nebenstehenden Generator. Ist α der Basiswinkel des aufgesetzten Dreiecks so gilt:

$$\cos\alpha = \frac{x}{r}$$

Da der Initiator die Einheitsstrecke ist, gilt $2r+2x=1$. Daraus folgt

$$\frac{1}{r} = 2(1+\cos\alpha)$$

Für einen Winkel von $\alpha = 86.35^\circ$, wie er sich in den unten angegebenen DATA-Werten ergibt, folgt

$$\frac{1}{r} = 2.1273 \Rightarrow D = \frac{\ln N}{\ln \frac{1}{r}} = 1.8365$$

Die fraktale Dimension ist somit $D = 1.8365$. Wirkt der Generator auf einem Quadrat, so erhält man das Caesàro-Viereck von Bild 12.5 mit der Generatorzeile

```
DATA 1,1,-1,1,-1,-1,1,-1,1,1
DATA .47,0,.5,.47,.53,0
```

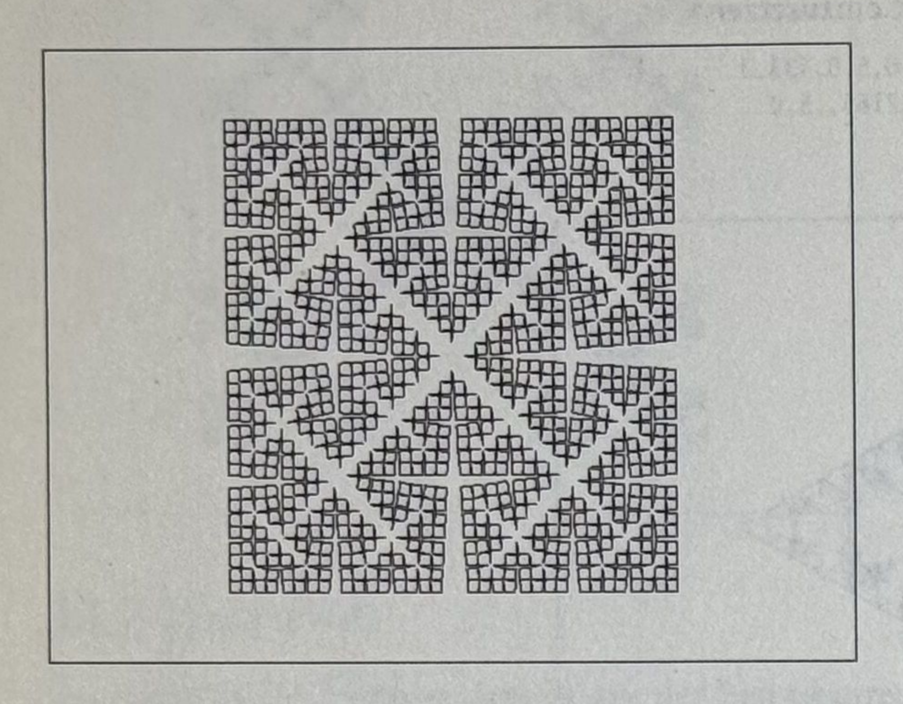


Bild 12.5 Caesàro-Quadrat

Interessant erscheint das Caesàro-Dreieck, wenn die Spitze nicht symmetrisch liegt, sondern die Strecke im Verhältnis 1:2 teilt. Das Innere der Abbildung 12.6 wurde zusätzlich schwarz gefärbt. Die Generatorzeile lautet hier

```
DATA 0,0,1,0
DATA .3,0,.3,.45,.35,0
```

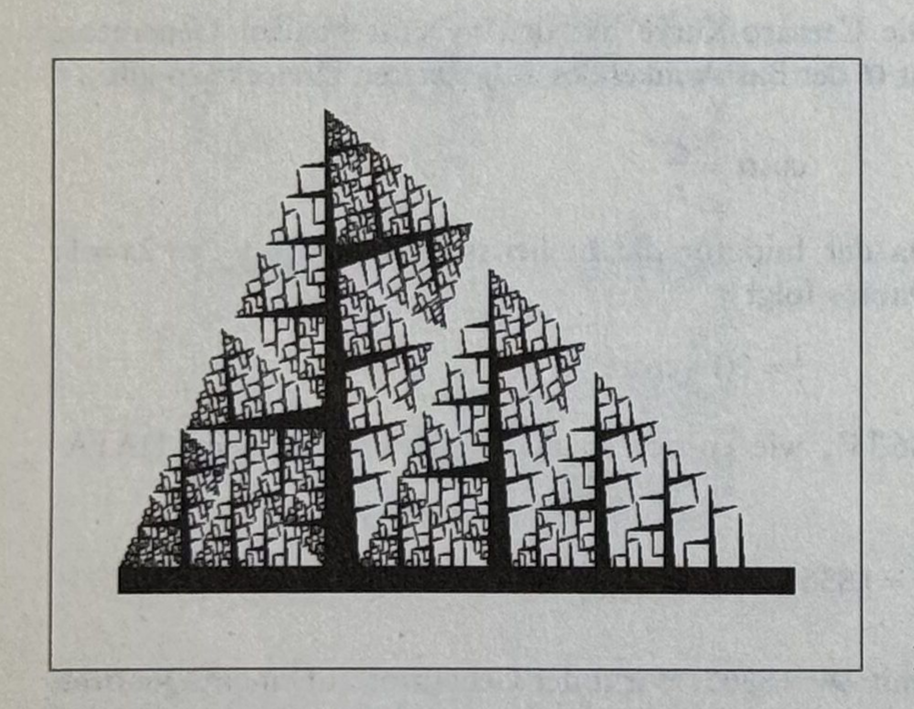
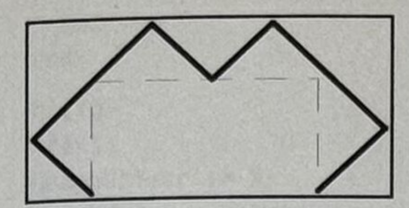


Bild 12.6 Caesaro-Dreieck

12.4 Lévy-Kurve



Die Kurve stammt von dem französischen Mathematiker Paul Lévy, der sich schon früh mit fraktalen Kurven befaßt hat. Der Generator 2. Stufe findet sich in nebenstehender Abbildung. Die fraktale Dimension ist nur schwer zu ermitteln, da es hier zu Selbstberührungen kommt.

Die Lévy-Kurve kann mit dem folgenden Basic-Programm `levy.bas` erzeugt werden:

```
'levy.bas

CONST PI = 3.141593
CONST p = 12: 'Rekursionstiefe
DIM a, b, m, n, s, x, y AS INTEGER
DIM h AS SINGLE

SCREEN 12: CLS
WINDOW (-.7, -1.1)-(1.7, .4)
LINE (-.7, -1.1)-(1.7, .4), , B
h = 2 ^ (-(p / 2))
a = h * COS(p * PI / 4): b = h * SIN(p * PI / 4)
LINE (0, 0)-(a, -b): LINE -(a + b, a - b)

x = 1: y = 1
FOR n = 2 TO 2 ^ p - 1
   m = n: s = 1
   DO
     IF m MOD 2 = 1 THEN s = s + 1
     m = m \ 2
   LOOP WHILE m > 1
   SELECT CASE s MOD 4
     CASE 0: x = x + 1
     CASE 1: y = y + 1
     CASE 2: x = x - 1
     CASE 3: y = y - 1
   END SELECT
   LINE -(a * x + b * y, a * y - b * x), 1 + s MOD 4
NEXT n
LOCATE 2, 2: PRINT "Levy-Kurve"
e$ = INPUT$(1): SCREEN 0
END
```

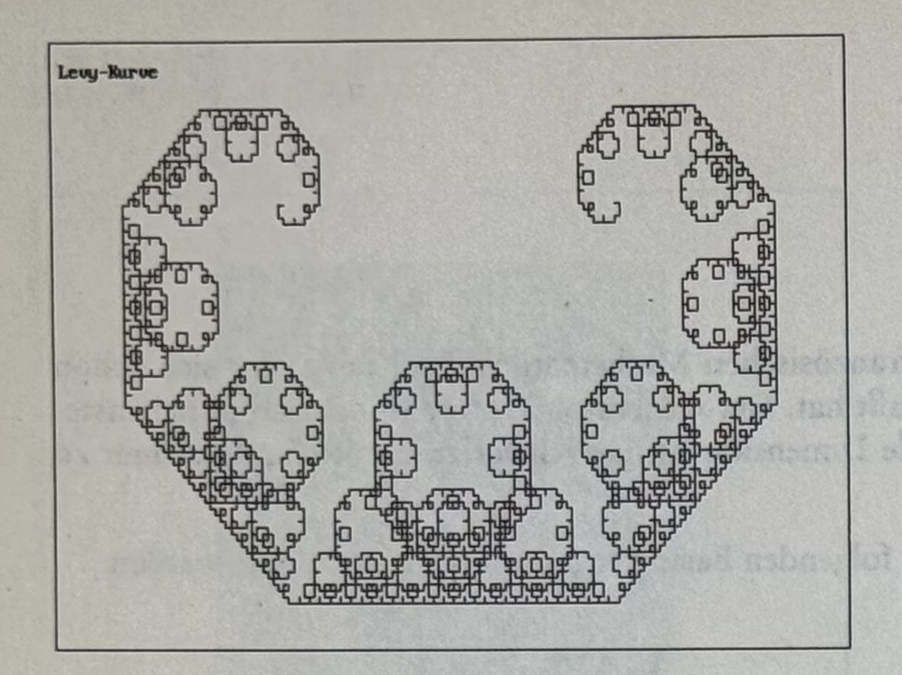


Bild 12.7 Lévy-Kurve

Mittels Turtle-Graphik läßt sich die Kurve in Turbo Pascal beschreiben.

```
program _levy; { Levy-Kurve }
uses crt,graph;

const startx = 100; starty = 330;
var turtx,turty,wi:real;
    Graphdriver,Graphmode:integer;

procedure forward(h:real);
var dx,dy,tx,ty:real;
begin
  dx := h*cos(wi*pi/180);
  dy := 2*h*sin(wi*pi/180);
  tx := turtx +dx; ty := turty +dy;
  line(round(2*turtx),round(turty),round(2*tx),round(ty));
  turtx := tx; turty := ty
end;

procedure turn(alpha:integer);
begin
  wi := wi+alpha
end;

procedure start;
begin
  wi := 0;
  turtx := startx; turty := starty;
end;
procedure levy(grad:integer;seite:real);
begin
if grad=0 then forward(seite)
else
  begin
```

```
  levy(grad-1,seite); turn(90);
  levy(grad-1,seite); turn(-90);
  end
end;
begin
start;
GraphDriver := 9;
GraphMode := VGAHi;
Initgraph(GraphDriver,GraphMode,'\tp\bgi');
SetgraphMode(Graphmode);
turn(180);
levy(12,2);
repeat until keypressed;
closegraph;
textmode(lastmode)
end.
```

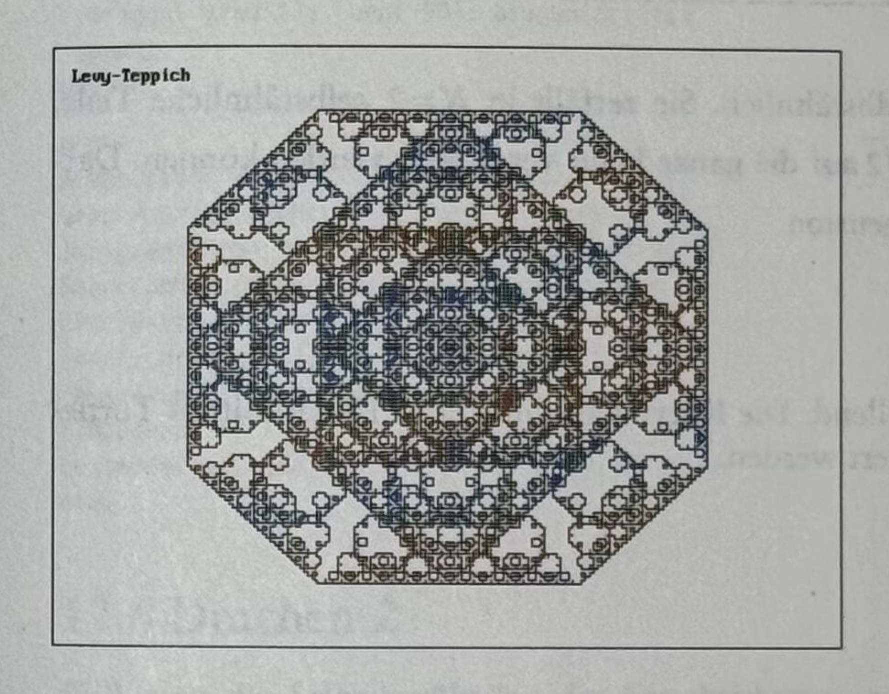


Bild 12.8 Levy-Teppich

Kombiniert man 4 Lévy-Kurven miteinander, die jeweils 90° gegeneinander gedreht sind, so erhält man den Lévy-Teppich (siehe Bild 12.8 und Tarbtafel 16). Er kann wieder aus dem Basic-Programm `koch.bas` erzeugt werden mittels des Generators

```
DATA 1,1,-1,1,-1,-1,1,-1,1,1 : 'Basis
DATA .5,.5 : 'Motiv
```

12.5 Drachenkurve

Die Drachenkurve wurde von den amerikanischen Mathematikern Harter und Heightway erfunden und durch eine Artikelserie im *Scientific American* vom März 1967 populär.

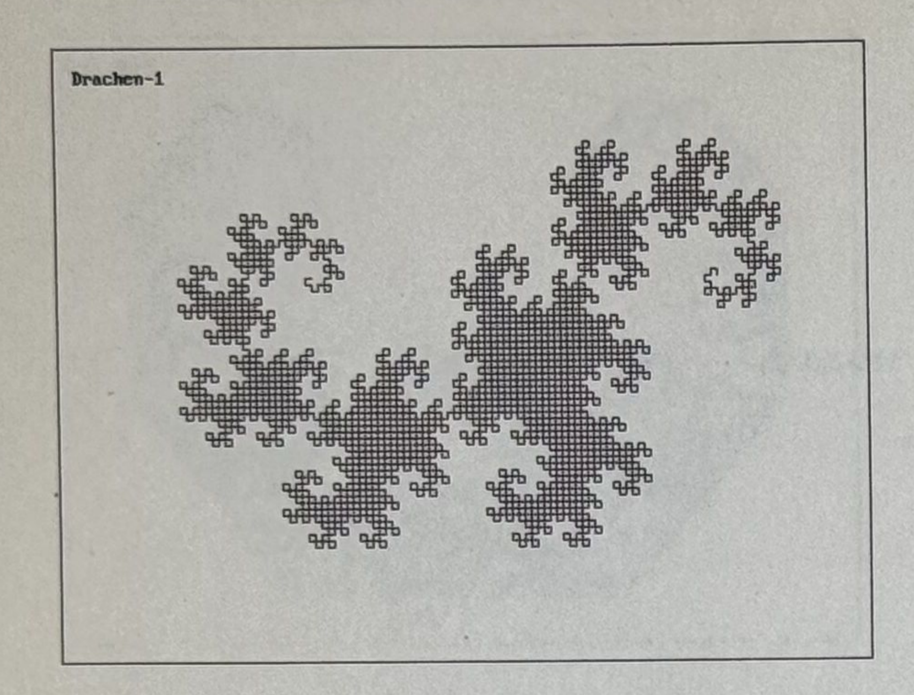


Bild 12.9 Drachenkurve

Die Drachenkurve ist strikt selbstähnlich. Sie zerfällt in $N = 2$ selbstähnliche Teile, die mit dem Streckfaktor $\frac{1}{r} = \sqrt{2}$ auf die ganze Figur vergrößert werden können. Daraus ergibt sich die fraktale Dimension

$$D = \frac{\ln N}{\ln \frac{1}{r}} = \frac{\ln 2}{\ln \sqrt{2}} = 2$$

Der Drachen ist also flächenfüllend. Die Kurve kann in Turbo Pascal mittels Turtle-Graphik, wie folgt, implementiert werden.

```
program _dragon; { Drachenkurve }
uses crt,graph;

const ds = 3;
      startx = 230; starty = 180;
var turtx,turty,wi:real;
    Graphdriver,Graphmode:integer;

procedure forward(h:real);
var dx,dy,tx,ty:real;
begin
  dx := h*cos(wi*pi/180);
  dy := 2*h*sin(wi*pi/180);
  tx := turtx +dx; ty := turty +dy;
  line(round(2*turtx),round(turty),round(2*tx),round(ty));
  turtx := tx; turty := ty
end;

procedure turn(alpha:integer);
begin
```

```
  wi := wi+alpha
end;

procedure start;
begin
  wi := 0;
  turtx := startx; turty := starty;
end;

procedure dragon(grad:integer);
begin
if grad=0 then forward(ds)
else if grad>0 then
  begin
  dragon(grad-1); turn(90); dragon(1-grad)
  end
else
  begin
  dragon(-grad-1); turn(-90); dragon(grad+1)
  end
end;

begin
GraphDriver := 9;
GraphMode := VGAHi;
Initgraph(GraphDriver,GraphMode,'\tp\bgi');
SetgraphMode(Graphmode);
cleardevice;
start; dragon(12);
repeat until keypressed;
closegraph;
textmode(lastmode)
end.
```

12.6 Drachen-2

Will man die Selbstberührung der Drachenkurve vermeiden, so müssen die Ecken abgerundet werden. Dies liefert den Drachen gemäß Bild 12.10. Zur Darstellung der gerundeten Drachenkurve dient das Basic-Programm `dragon2.bas`

```
'dragon2.bas
'Drachenkurve mit gerundeten Ecken
CONST p = 10: 'Rekursionstiefe
CONST PI = 3.141593
DIM d, n, s AS INTEGER
DIM h, x, x1, x2, x3, y, y1, y2, y3 AS SINGLE

SCREEN 12: CLS :
WINDOW (-.5, -.9)-(1.5, .5)
LINE (-.5, -.9)-(1.5, .5), , B
```

```
h = 2 ^ (-p / 2): s = 0
x = h * COS(p * PI / 4): y = -h * SIN(p * PI / 4)
LINE (0, 0)-(.75 * x, .75 * y), 1
FOR n = 1 TO 2 ^ p - 1
    m = n
    WHILE m MOD 2 = 0
       m = m / 2
    WEND
    IF m MOD 4 = 1 THEN d = 1 ELSE d = -1
    s = (s + d) MOD 4
    x1 = x + h * COS((s - p / 2) * PI / 2)
    y1 = y + h * SIN((s - p / 2) * PI / 2)
    x2 = (3 * x + x1) / 4: y2 = (3 * y + y1) / 4
    x3 = (x + 3 * x1) / 4: y3 = (y + 3 * y1) / 4
    LINE -(x2, y2), 1 + n MOD 7
    LINE -(x3, y3), 1 + n MOD 7
    x = x1: y = y1
NEXT n
LINE -(1, 0), 14
LOCATE 2, 5: PRINT "Drachen-2"
e$ = INPUT$(1): SCREEN 0
END
```

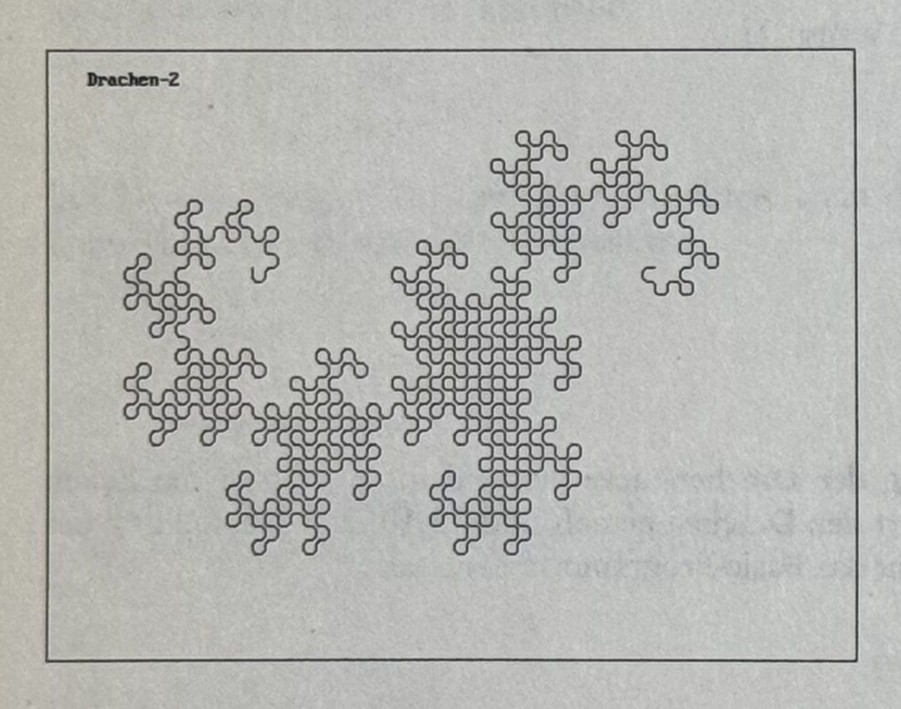


Bild 12.10 Gerundete Drachenkurve

12.7 Cayley-Baum

Der Cayley-Baum ist eine Abart der bekannten H-Kurve; er wird manchmal auch nach Mandelbrot benannt. Der Cayley-Baum kann mit dem Quick Basic-Programm `cayley.bas` gezeichnet werden.

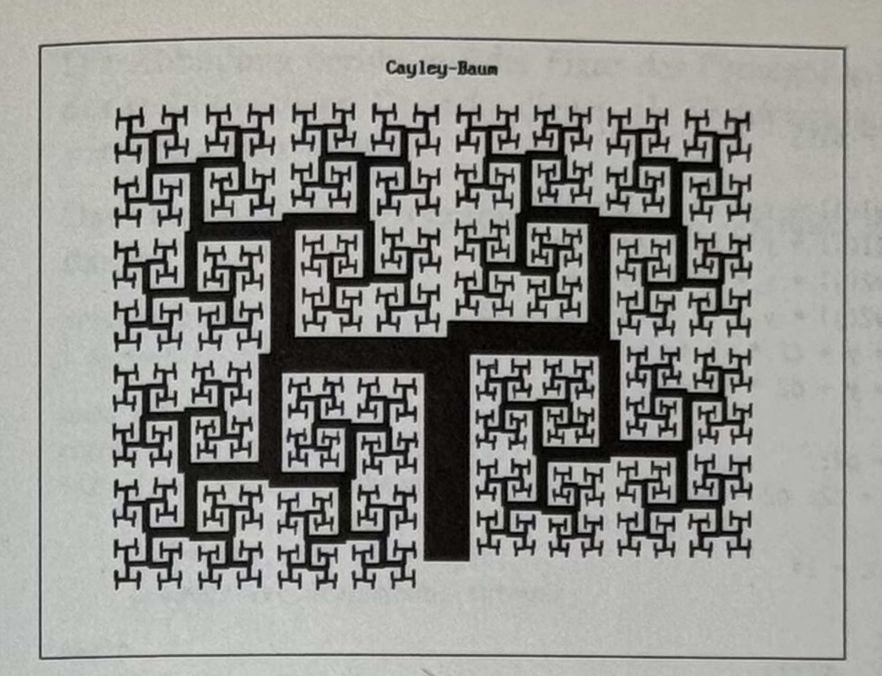


Bild 12.11 Cayley-Baum

```
'cayley.bas
'Cayley-Baum

CONST p = 11
CONST r1 = .72: r2 = .67: a = 3.98: b = 4.38
DIM c, j, n, m, s AS INTEGER
DIM a1, a2, b1, b2, c1, c2, d1, d2, f1, f2, h, x, y AS SINGLE
DIM x1(p), y1(p), x2(p), y2(p), u1(p), v1(p), u2(p), v2(p)

SCREEN 12: CLS
WINDOW (-9.5, -3)-(10.5, 12)
a1 = 0: a2 = a: b1 = 0: b2 = a + r1
e1 = 1: e2 = b + r2: f1 = 1: f2 = b
c1 = .5: c2 = b2: d1 = .5: d2 = e2
x1(0) = 0: y1(0) = 0: u1(0) = 1: v1(0) = 0
LINE (0, 0)-(1, 0), 10

s = 1
GOSUB Schleife
FOR m = 1 TO 2 ^ (p - 1) - 1
  s = p: n = m
  WHILE n MOD 2 = 0
     n = n \ 2: s = s - 1
  WEND
  h = a2: a2 = f2: f2 = h: h = b2: b2 = e2
  e2 = h: h = c2: c2 = d2: d2 = h
  x1(s - 1) = x2(s - 1): y1(s - 1) = y2(s - 1)
  u1(s - 1) = u2(s - 1): v1(s - 1) = v2(s - 1)
  GOSUB Schleife
NEXT m
LOCATE 1, 35: PRINT "Cayley-Baum"
a$ = INPUT$(1): SCREEN 0
END
```

```
Schleife:
FOR j = s TO p
 x = x1(j - 1): y = y1(j - 1): u = u1(j - 1): v = v1(j - 1)
 x3 = u - x: y3 = v - y
 x1(j) = x + a1 * x3 - a2 * y3: y1(j) = y + a2 * x3 - a1 * y3
 u1(j) = x + b1 * x3 - b2 * y3: v1(j) = y + b2 * x3 + b1 * y3
 x2(j) = x + e1 * x3 - e2 * y3: y2(j) = y + e2 * x3 + e1 * y3
 u2(j) = x + f1 * x3 - f2 * y3: v2(j) = y + f2 * x3 + f1 * y3
 u3 = x + c1 * x3 - c2 * y3: v3 = y + c2 * x3 + c1 * y3
 u4 = x + d1 * x3 - d2 * y3: v4 = y + d2 * x3 + d1 * y3
 IF j = s THEN
     h = a2: a2 = f2: f2 = h: h = b2:
     b2 = e2: e2 = h: h = c2: c2 = d2: d2 = h
  END IF
  c = 1 + j MOD 3: IF c = 3 THEN c = 14
  LINE (x, y)-(x1(j), y1(j)), c
  LINE (u1(j), v1(j))-(u3, v3), c
  LINE -(u4, v4), c: LINE -(x2(j), y2(j)), c
  LINE (u2(j), v2(j))-(u, v), c
NEXT j
RETURN
```

12.8 Symmetrischer Pythagoras-Baum

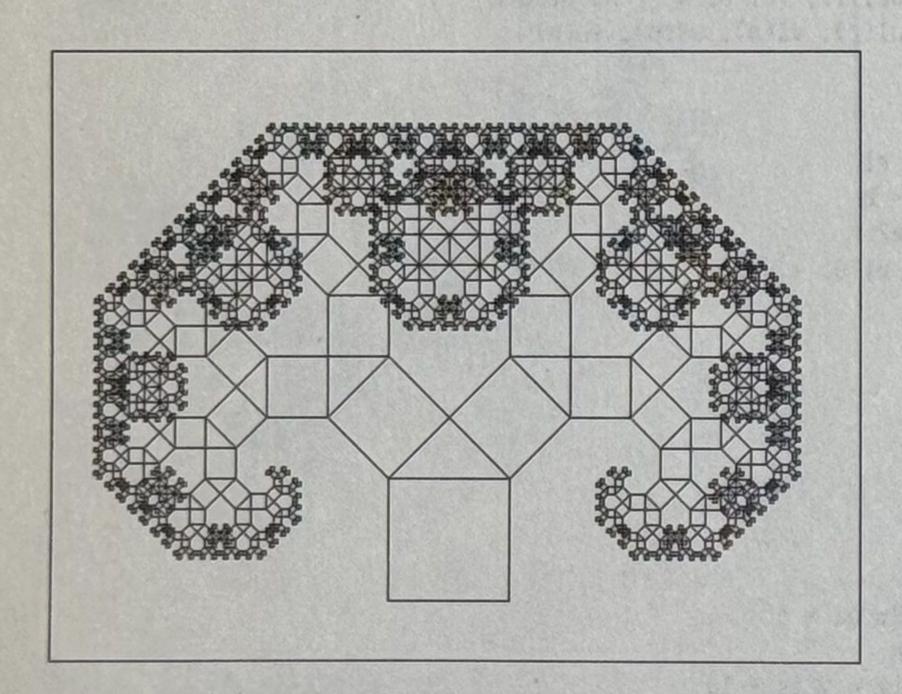


Bild 12.12 Symmetrischer Pythagoras-Baum

Die bekannten Pythagoras-Bäume gehen zurück auf den holländischen Ingenieur A. E. Bosman. Er entwarf diese während des 2.Weltkriegs an dem Zeichenbrett, an dem er sonst seine U-Boot-Pläne zeichnete. Seine Graphiken wurden später in dem Buch *Het wondere onderzoekingsveld der vlakke meetkunde* (1957) publiziert.

Die Abbildung beruht auf der Figur des Pythagoras-Lehrsatzes. Die beiden Katheten des rechtwinkligen Dreiecks dienen als Verzweigung, da jedes Kathetenquadrat sich wiederum verzweigt.

Das Turbo Pascal-Programm `pythbl.pas` zeichnet den symmetrischen Pythagoras-Baum.

```
program pythbl;
{ symmetrischer Pythagorasbaum }

uses crt, graph;
const c = 0.707107; { 1/sqrt(2)}
var  j,k,l,m,n,p : integer;
     f,h,u,v,x,y : real;
     d:array[1..10] of integer;
     Graphdriver,Graphmode: integer;

begin
GraphDriver := Detect;
Initgraph(GraphDriver,GraphMode,' ');
SetgraphMode(Graphmode);
cleardevice;

x := 0; y := 0; u := 1; v := 1;
for m := 0 to 10 do
  begin
  p := 1;
  for k:= 1 to m do p := p*2;
  for n := p to 2*p - 1 do
    begin
    l := n; h := 1;
    x := 0; y := 0; f := 0;
    for k := 0 to m-1 do
      begin
      d[m - k] := l mod 2; l := l div 2
      end;
    x := 0; y := 0;
    for j := 1 to m do
    if d[j] = 0 then
      begin
      x := x - h * (cos(f) + 2 * sin(f));
      y := y + h * (2 * cos(f) - sin(f));
      f := f + pi / 4; h := c * h
      end
     else
       begin
       x := x + h * (cos(f) - 2 * sin(f));
       y := y + h * (2 * cos(f) + sin(f));
       f := f - pi / 4; h := c * h
       end;
    u := h * (cos(f) + sin(f));
    v := h * (cos(f) - sin(f));
```

```
    setcolor(m+4);
    moveto(round(48*(x-v+6.5)),480-round(48*(y-u+2)));
    lineto(round(48*(x+u+6.5)),480-round(48*(y-v+2)));
    lineto(round(48*(x+v+6.5)),480-round(48*(y+u+2)));
    lineto(round(48*(x-u+6.5)),480-round(48*(y+v+2)));
    lineto(round(48*(x-v+6.5)),480-round(48*(y-u+2)))
   end
end;
repeat until keypressed;
textmode(lastmode)
end.
```

12.9 Schiefer Pythagorasbaum

Beim symmetrischen Pythagoras-Baum schließen die Katheten mit der Hypothenuse einen 45° Winkel ein. Ändert man diesen Winkel, so wird der Baum schief.

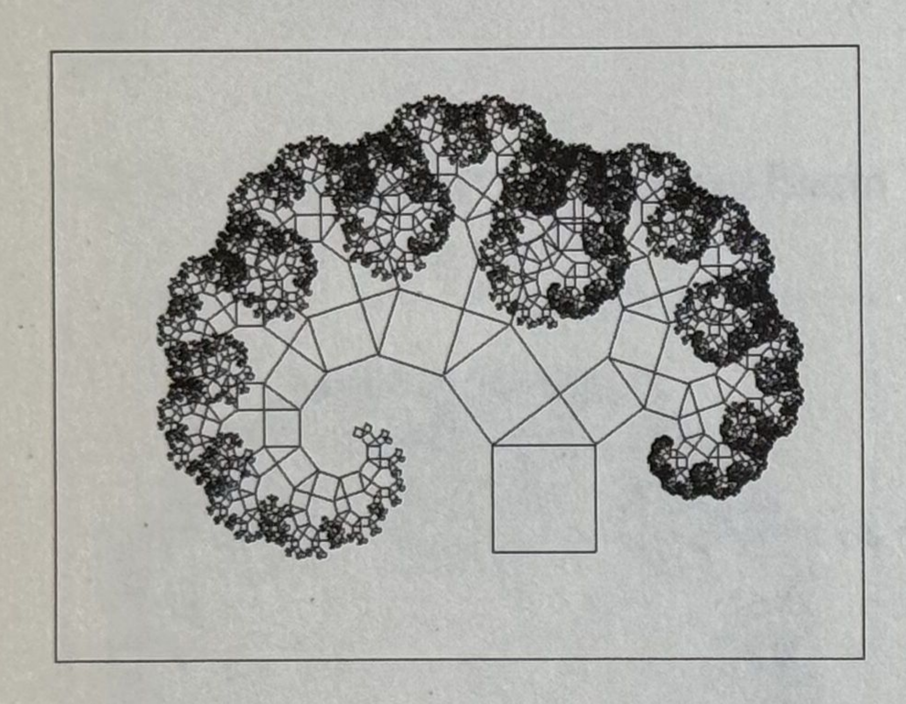


Bild 12.13 Schiefer Pythagorasbaum

Zum Zeichnen des schiefen Pythagoras-Baums dient das Quick Basic-Programm pythb2.bas.

```
'pythb2.bas
'schiefer Pythagorasbaum

CONST PI = 3.141593
CONST p = 12: 'Rekursionstiefe
DIM j, m, n, s AS INTEGER
DIM c, f, a1, a2, b1, b2, c1, c2, d1, d2 AS SINGLE
DIM x1(p), y1(p), x2(p), y2(p), u1(p), v1(p), u2(p), v2(p)
```

```
SCREEN 12: CLS
WINDOW (-4.2, -2)-(3.6, 3.7)

f = PI / 6: 'Winkel
c = COS(f): si = SIN(f)
a1 = -c * si: a2 = c ^ 2
b1 = a1 + a2: b2 = -a1 + a2
c1 = b2: c2 = 1 - b1
d1 = 1 - a1: d2 = 1 - a2
x1(0) = 0: y1(0) = 0
u1(0) = 1: v1(0) = 0
LINE (0, 0)-(0, -1), 3: LINE -(1, -1), 3
LINE -(1, 0), 3
s = 1
GOSUB graph
FOR m = 1 TO 2 ^ (p - 1) - 1
  s = p: n = m
  WHILE n MOD 2 = 0
     n = n \ 2: s = s - 1
  WEND
  x1(s - 1) = x2(s - 1): y1(s - 1) = y2(s - 1)
  u1(s - 1) = u2(s - 1): v1(s - 1) = v2(s - 1)
  GOSUB graph
NEXT m
e$ = INPUT$(1): SCREEN 0
END

graph:
FOR j = s TO p
X = x1(j - 1): Y = y1(j - 1): U = u1(j - 1): V = v1(j - 1)
X3 = U - X: Y3 = V - Y
x1(j) = X + a1 * X3 - a2 * Y3: y1(j) = Y + a2 * X3 + a1 * Y3
u1(j) = X + b1 * X3 - b2 * Y3: v1(j) = Y + b2 * X3 + b1 * Y3
x2(j) = X + c1 * X3 - c2 * Y3: y2(j) = Y + c2 * X3 + c1 * Y3
u2(j) = X + d1 * X3 - d2 * Y3: v2(j) = Y + d2 * X3 + d1 * Y3
LINE (X, Y)-(x1(j), y1(j)), s + 2
LINE -(u1(j), v1(j)), s + 2: LINE -(U, V), s + 2
LINE -(X, Y), s + 2: LINE -(x2(j), y2(j)), s + 2
LINE -(u2(j), v2(j)), s + 2: LINE -(U, V), s + 2
NEXT j
RETURN
```

Setzt man hier den Winkel $f = \frac{\pi}{4}$, so erhält man wieder den symmetrischen Pythagoras-Baum.

12.10 Alternierender Pythagoras-Baum

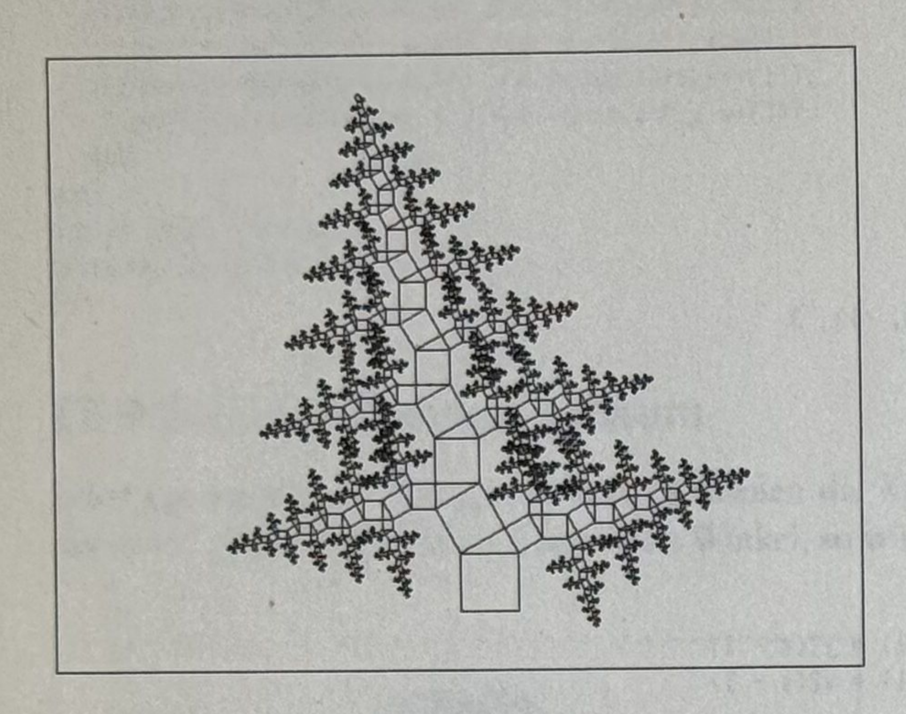


Bild 12.14 Alternierender Pythagoras-Baum

Ebenfalls von Bosmann stammt die Idee für den alternierenden Pythagoras-Baum. Dies erreicht man, indem man bei jeder Verzweigung die beiden Kathetenquadrate vertauscht; d.h. das jeweils kleinere Quadrat wird abwechselnd links und rechts aufgetragen (Bild 12.14 und Farbtafel 11).

Dieser Pythagoras-Baum ist nicht einfach zu programmieren. Das folgende Basic-Programm `pythb3.bas` ist daher etwas umfangreicher.

```
'pythb3.bas
'alternierender Pythagorasbaum

CONST TRUE = 1: p = 10: eps = .05
DIM j, k, m AS INTEGER
DIM a, b, c, f, x, y AS SINGLE
DIM a1, a2, a3, a4, b1, b2, b3, b4 AS SINGLE
DIM c1, c2, c3, c4, d1, d2, d3, d4 AS SINGLE
DIM u, v, q, u1, v1, x0, x1, y0, y1 AS SINGLE
DIM x2(p), y2(p), x3(p), y3(p), x4(p), y4(p)
DIM u2(p), v2(p), u3(p), v3(p), u4(p), v4(p), s(p)

SCREEN 12: CLS
WINDOW (-7, -1)-(7, 9.5)

f = .5: a = COS(f) ^ 2
b = SIN(f) * COS(f): c = SIN(f) ^ 2
a1 = -b: b1 = 1 + a: c1 = -b: d1 = 1 + a + b
a2 = c1: b2 = d1: c2 = a - b: d2 = 1 + a + b
a3 = a + b: b3 = 1 + b + c: c3 = 1 + b: d3 = 1 + b + c
a4 = c3: b4 = d3: c4 = 1 + b: d4 = 1 + c
```

```
x = 0: y = 0: u = 1: v = 0
q = 0: s(0) = 3: j = 1

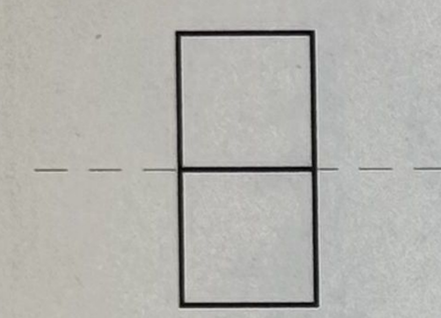
WHILE TRUE: 'Abbruch durch Backtracking
m = q + j: x0 = u - x: y0 = v - y
x1 = x + a1 * x0 - b1 * y0: y1 = y + b1 * x0 + a1 * y0
u1 = x + c1 * x0 - d1 * y0: v1 = y + d1 * x0 + c1 * y0
x2(m) = x + a2 * x0 - b2 * y0: y2(m) = y + b2 * x0 + a2 * y0
u2(m) = x + c2 * x0 - d2 * y0: v2(m) = y + d2 * x0 + c2 * y0
x3(m) = x + a3 * x0 - b3 * y0: y3(m) = y + b3 * x0 + a3 * y0
u3(m) = x + c3 * x0 - d3 * y0: v3(m) = y + d3 * x0 + c3 * y0
x4(m) = x + a4 * x0 - b4 * y0: y4(m) = y + b4 * x0 + a4 * y0
u4(m) = x + c4 * x0 - d4 * y0: v4(m) = y + d4 * x0 + c4 * y0
s = ABS(x - u) + ABS(y - v): s(m) = 3
GOSUB graphik
x = x1: y = y1: u = u1: v = v1
IF m = p OR s < eps THEN GOSUB backtrack
j = j + 1
WEND

graphik:
LINE (x + y - v, y - x + u)-(x1, y1), k + 1
LINE -(u2(m), v2(m)), k + 1
LINE -(u - v + y, u + v - x), k + 1
LINE -(u4(m), v4(m)), k + 1
LINE -(x3(m), y3(m)), k + 1
LINE -(x + y - v, y - x + u), k + 1
LINE -(x, y), k + 1: LINE -(u, v), k + 1
LINE -(u - v + y, u + v - x), k + 1
LINE -(x + y - v, y - x + u), k + 1
RETURN

backtrack:
k = 1
WHILE s(m - k) = 0
     k = k + 1
WEND
IF m <> k THEN
   q = m - k: x = x2(q): y = y2(q): u = u2(q): v = v2(q)
   x2(q) = x3(q): y2(q) = y3(q): u2(q) = u3(q): v2(q) = v3(q)
   x3(q) = x4(q): y3(q) = y4(q): u3(q) = u4(q): v3(q) = v4(q)
   s(q) = s(q) - 1: j = 0
   RETURN
ELSE
   e$ = INPUT$(1): SCREEN 0
   END
END IF
```

12.11 Peano-Kurve

Die Peano-Kurve hat den nebenstehenden Generator. Der Initiator ist eine Strecke, der Generator errichtet über dem mittleren Drittel je ein Quadrat nach oben und unten.



Damit ergibt die Drittelung der Strecke $\frac{1}{r}=3$ insgesamt $N=9$ Teilstrecken. Die fraktale Dimension berechnet sich daraus zu

$$D=\frac{\ln 9}{\ln 3}=2$$

Die Kurve ist daher flächenfüllend.

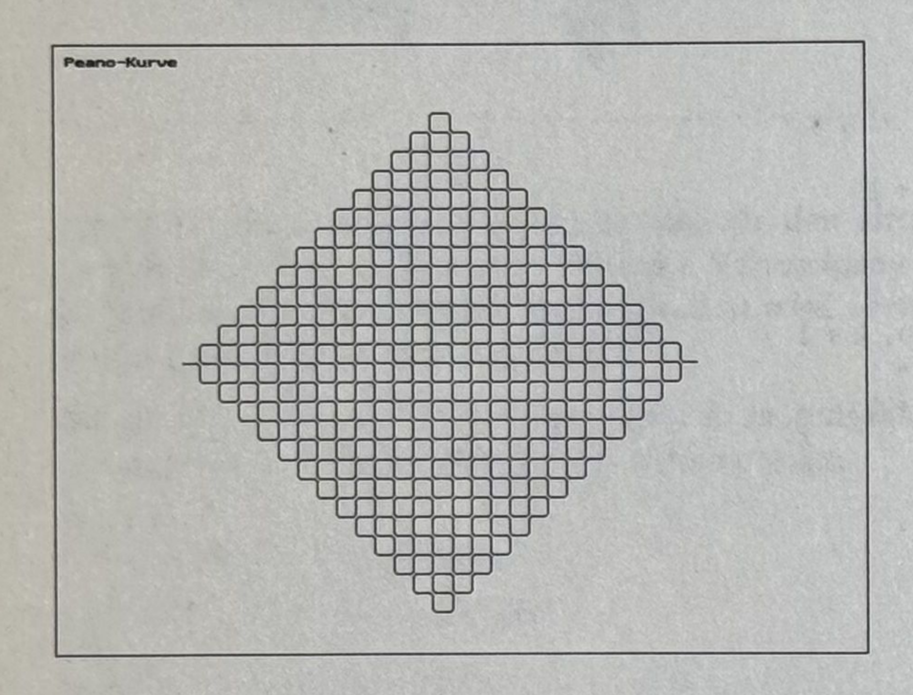


Bild 12.15 Peano-Kurve

Der Generator kann direkt in Turtle-Graphik umgesetzt werden, indem man die Prozedur `peano` entsprechend definiert. Bei jedem Aufruf muß die Seite gedrittelt werden, das Rückzählen der Rekursionsstufe garantiert das Abbrechen der Rekursion. Dies zeigt das Pascal-Programm `peano.pas`.

```
program _peano; { Peano-Kurve mittels Turtle-Graphik }
uses crt,graph;

const ds = 1; startx = 50; starty = 250;
var turtx,turty,wi:real;
    Graphdriver,Graphmode:integer;

procedure forward(h:real);
```

```
var dx,dy,tx,ty:real;
begin
  dx := h*cos(wi*pi/180);
  dy := 2*h*sin(wi*pi/180);
  tx := turtx +dx; ty := turty +dy;
  line(round(2*turtx),round(turty),round(2*tx),round(ty));
  turtx := tx; turty := ty
end;

procedure turn(alpha:integer);
begin
  wi := wi+alpha
end;

procedure start;
begin
  wi := 0; turtx := startx; turty := starty;
end;

procedure left;
begin
turn(45); forward(ds*sqrt(2)); turn(45)
end;

procedure right;
begin
turn(-45); forward(ds*sqrt(2)); turn(-45)
end;

procedure peano(grad:integer;seite:real);
begin
if grad=0 then forward(seite)
else
  begin
  peano(grad-1,seite/3); left;
  peano(grad-1,seite/3); right;
  peano(grad-1,seite/3); right;
  peano(grad-1,seite/3); right;
  peano(grad-1,seite/3); left;
  peano(grad-1,seite/3); left;
  peano(grad-1,seite/3); left;
  peano(grad-1,seite/3); right;
  peano(grad-1,seite/3)
  end
end;

begin { main }
start;
GraphDriver := 9;
GraphMode := VGAhi;
Initgraph(GraphDriver,GraphMode,'\tp\bgi');
SetgraphMode(Graphmode);
peano(3,150);
```

```
repeat until keypressed;
closegraph;
textmode(lastmode)
end.
```



13 Iterierte Funktionssysteme (IFS)

Als Michael Barnsley auf der SIGGRAPH ´85 sein Verfahren zur graphischen Erzeugung des Farnkrauts mit Hilfe des von ihm entwickelten Iterierten Funktionen-Systems (IFS) darstellte, erregte er großes Aufsehen.

Es war völlig neuartig, daß ein so kompliziertes pflanzliches Muster nur mit Hilfe von 4 affinen Abbildungen dargestellt werden konnte!

13.1 IFS-Code des Farns

Eine affine Abbildung kann dargestellt werden in der Form

$$\begin{pmatrix} x' \\ y' \end{pmatrix} = \begin{pmatrix} a & b \\ c & d \end{pmatrix} \cdot \begin{pmatrix} x \\ y \end{pmatrix} + \begin{pmatrix} e \\ f \end{pmatrix}$$

dabei wird in der Regel gefordert, daß die Determinante der 2-reihigen Matrix nicht verschwindet. In einzelnen Fällen ergeben sich jedoch auch Transformationen mit verschwindender Determinante. Einfacher als die Matrixform ist die Darstellung eines IFS mittels Tabelle. Tabelle 1 zeigt den IFS-Code für Farn

a	b	c	d	e	f
0	0	0	0.16	0	0
0.197	-0.026	0.226	0.197	0	1.6
-0.155	0.283	0.26	0.237	0	0.44
0.849	0.037	-0.037	0.849	0	1.6

Tab. 1 IFS-Code des Barnsley-Farn

Die verwendeten affinen Abbildungen sind in Bild 13.1 dargestellt. Abbildung I aus Zeile 1 (mit verschwindender Determinante) stellt eine Projektion des Einheitsquadrats auf die Strecke [(0/0),(0.16)] dar und liefert den Stengel des Farnkrauts.

Die Abbildungen II und IV (aus Zeile 2 bzw. 4) stellen Ähnlichkeitsabbildungen mit den Streckfaktoren 0.3 bzw. 0.85 dar. Die Abbildung III ist eine Affinität mit der Determinante -0.106.

Die Abbildungen II und III erzeugen die Blätter des Farns; die Abbildung IV bewirkt mittels einer Drehung um -2.5 Grad die gekrümmte Form des Farns.

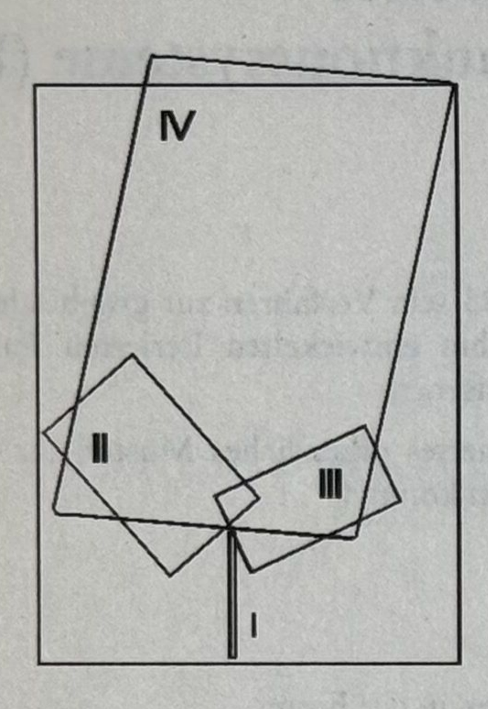


Bild 13.1 Die 4 affinen Abbildungen des Farns

Die einfachste Methode zur Erzeugung einer Figur nach der IFS-Methode ist der stochastische Algorithmus nach Barnsley. Dabei werden mehrere tausend Punkte den affinen Abbildungen unterworfen; die jeweils ausgeführte Abbildung wird mittels Zufallszahlen ausgelost. Als Punkte werden jeweils die zuletzt erhaltenen Bildpunkte gewählt, beim Start der Nullpunkt.

Der stochastische Algorithmus kann durch das Turbo Pascal-Programm `farn.pas` ausgeführt werden:

```
program farn;
{ Stochastischer IFS-Algorithmus
  nach Michael Barnsley: Fractals Everywhere
  Academic Press, San Diego 1988, p.91 }
uses crt,graph;

const
 a:array[1..4] of real = (0,0.197,-0.155,0.849);
 b:array[1..4] of real = (0,-0.226,0.283,0.037);
 c:array[1..4] of real = (0,0.226,0.26,-0.037);
 d:array[1..4] of real = (0.16,0.197,0.237,0.849);
 e:array[1..4] of real = (0,0,0,0);
```

```
 f:array[1..4] of real = (0,1.6,0.44,1.6);
 p:array[1..4] of real = (0.03,0.14,0.27,1.0);
var  k : integer;
     i:longint;
     x,x1,y,pk : real;

procedure GraphInit;
var  Graphdriver,Graphmode:integer;
begin
GraphDriver := Detect;
Initgraph(GraphDriver,GraphMode,'\tp\bgi');
SetgraphMode(Graphmode);
cleardevice;
end;

begin
GraphInit;
randomize;
x := 0; y := 0;
for i:= 1 to 75000 do
   begin
   pk := random;
   if pk <= p[1] then k := 1
   else if pk <= p[2] then k := 2
   else if pk <= p[3] then k := 3
   else k := 4;
   x1 := a[k] * x + b[k] * y + e[k];
   y := c[k] * x + d[k] * y + f[k];
   x := x1;
   putpixel(round(x*50)+320,430-round(y*40),10);
end;

repeat until keypressed;
closegraph;
textmode(lastmode)
end.
```

Der IFS-Code ist bei obigem Programm in Form von typisierten Konstanten gegeben. Zu beachten ist, daß die Wahrscheinlichkeiten über ihre Verteilungsfunktion eingegeben werden müssen.

Für die Wahl der Wahrscheinlichkeiten p gibt Barnsley [02] (S.87) folgende Faustregel an. Die Wahrscheinlichkeit p, mit der eine Abbildung ausgeführt werden soll, wird näherungsweise gleich dem Verhältnis aus der Determinante der Abbildung zur Summe der Determinanten aller Abbildungen gesetzt.

$$p \approx \frac{|a_i d_i - b_i c_i|}{\sum_i |a_i d_i - b_i c_i|}$$

13.2 Variationen des Farn-Codes

Durch Variationen des Farn-IFS-Codes kann man zahlreiche baumähnlichen Strukturen finden. Der IFS-Code für viele weitere Figuren wird in Tabellenform gegeben.



Bild 13.2 Farn nach Barnsley

Die vollständigen Pascal-Programme dazu – einschließlich der hier nicht angegebenen Wahrscheinlichkeiten – finden sich auf der beigefügten Diskette unter der jeweiligen Tabellennummer.

a	b	c	d	e	f
0	0	0	0.2	0	0
0.2	-0.3	0.23	-0.05	0	0.5
-0.05	0.23	-0.3	-0.2	0	0.5
0.85	0	0	0.85	0	0.7

Tab. 2 Farn2

a	b	c	d	e	f
0	0	0	0.16	0	0
0.2	-0.26	0.23	0.22	0	0.5
-0.15	0.28	0.26	0.24	0	0.5
0.85	0	0	0.85	0	0.7

Tab. 3 Farn3

a	b	c	d	e	f
0	0	0	0.16	0	0
0.2	0.26	0.23	-0.22	0	1.6
0.15	0.28	-0.26	0.24	0	0.44
-0.85	0.04	0.04	0.85	0	1.6

Tab. 4 Farn4

a	b	c	d	e	f
0	0	0	0.16	0	0
0.2	-0.26	0	0.22	0	1.6
0.15	0.28	0	0.24	0	0.44
0.85	0.04	0	0.85	0	1.6

Tab. 5 Baum1

a	b	c	d	e	f
0	0	0	0.16	0	0
0.2	-0.26	0	0.22	0	1.6
0.65	0.28	0	0.24	0	0.44
-0.85	0.04	0	0.85	0	1.6

Tab. 6 Baum2

a	b	c	d	e	f
0.81	-0.017	0.017	0.8	0.087	0.182
-0.095	-0.385	-0.186	0.2	0.51	0.276
-0.085	0.397	0.23	0.13	0.5	0.
0	0.	0	0.227	0.46	0.017

Tab. 7 Baum3

a	b	c	d	e	f
0.8	0	0	0.85	0.09	0.18
-0.1	0.2	-0.4	0.2	0.51	0.28
-0.09	-0.2	0.4	0.13	0.5	0
0	0	0	0.2	0.46	0.02

Tab. 8 Baum4

a	b	c	d	e	f
0.75	0	0	0.8	0	0.2
0.33	0.3	-0.3	0.3	0	0.2
0.33	-0.2	0.5	0.2	0	0.2
0	0	0	0.3	0	0

Tab. 9 Baum5

13.3 Weitere IFS-Codes für Bäume

Ein Beispiel eines einfachen Zweiges ergibt sich aus den 3 affinen Abbildungen

a	b	c	d	e	f
-0.467	0.02	-0.113	0.015	0.4	0.4
0.387	0.43	0.43	-0.387	0.256	0.522
0.441	-0.091	-0.009	-0.322	0.422	0.506

Tab. 10 Zweig1

Schwieriger zu ermitteln ist die Darstellung eines gekrümmten Zweigs, erzeugt aus 5 Affinitäten

a	b	c	d	e	f
0.02	-0.05	0	0.29	0.65	0.38
0.35	0.26	-0.22	0.41	0.14	0.53
-0.05	-0.13	0	-0.47	0.74	1.03
0.029	-0.25	0.18	0.4	0.7	-0.07
0.48	0	0	0.5	0.26	0.14

Tab. 11 Zweig 2

Aus solchen einfachen Zweigen lassen sich komplexere Bilder erzeugen.

Ein besonders schöner Baum ist durch Tab. 12 gegeben; hier benötigt man allerdings acht affine Abbildungen.

a	b	c	d	e	f
0.008574	-0.12698	0.00535	0.20347	0.52394	0.16848
0.4481	0.52868	-0.21712	0.53388	0.293427	0.49045
0.0118	-0.00223	0.00007	0.37293	0.51142	0.01948
-0.04474	-0.41931	0.43653	-0.04297	0.48088	0.09414
0.63136	-0.13708	0.13026	0.66438	0.10622	0.23675
0.27107	0.34582	-0.2493	0.37607	0.36307	0.35753
-0.11509	0.29652	-0.21926	-0.1556	0.62457	0.43811
0.01649	0	0	0.06127	0.50875	0.00221

Tab. 12 Baum 6

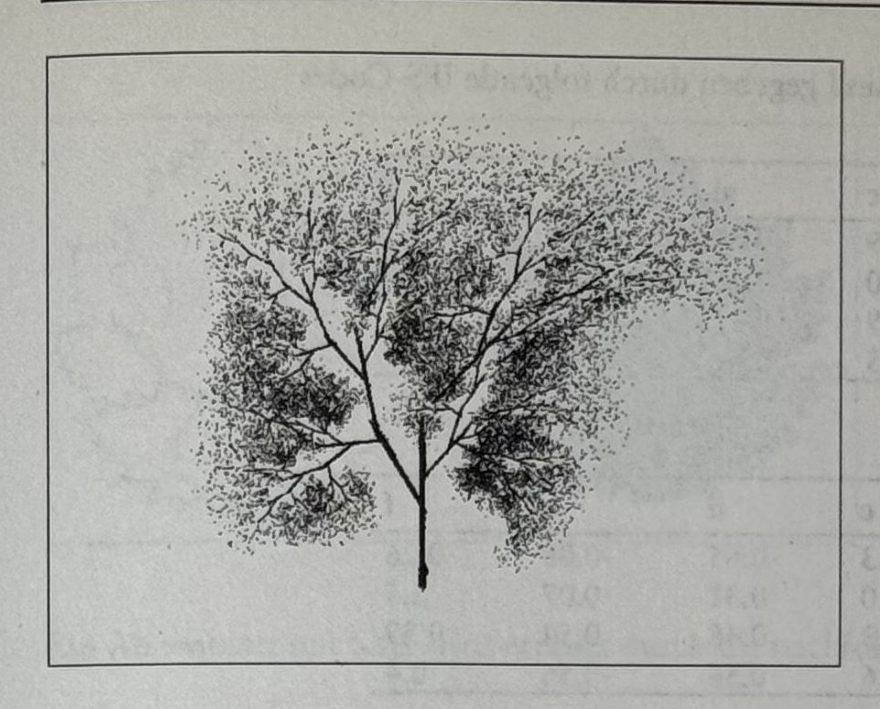


Bild 13.3 Baum nach Tab. 12

Nicht so kompliziert ist das Bild einer knorrigen alten Eiche, erzeugt durch die 5 Affinitäten

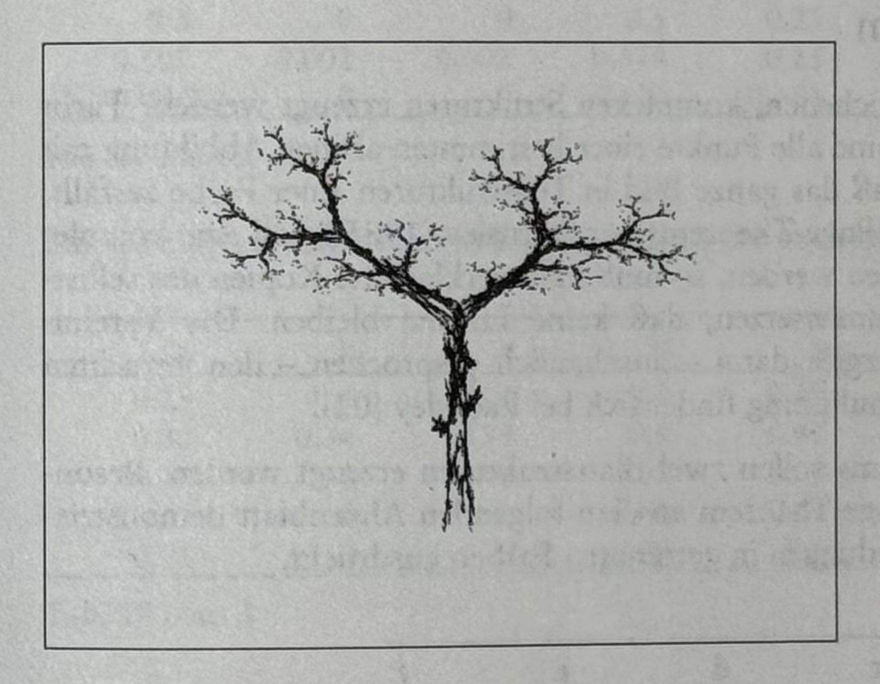


Bild 13.4 Eiche nach Tab. 13

a	b	c	d	e	f
0.195	-0.488	0.344	0.443	0.4431	0.2452
0.462	0.414	-0.252	0.361	0.2511	0.5692
-0.058	-0.07	0.453	-0.111	0.5976	0.0969
-0.035	0.07	-0.469	-0.022	0.4884	0.5069
-0.637	0	0	0.501	0.8562	0.2513

Tab. 13 Eiche

Zwei kiefern-ähnliche Bäume sind gegeben durch folgende IFS-Codes

a	b	c	d	e	f
-0.04	0	-0.19	-0.47	-0.12	0.3
0.65	0	0	0.56	0.06	1.56
0.41	0.46	-0.39	0.61	0.46	0.4
0.52	-0.35	0.25	0.74	-0.48	0.38

Tab. 14 Baum 7

a	b	c	d	e	f
-0.04	0	-0.23	-0.65	-0.08	0.26
0.61	0	0	0.31	0.07	2.5
0.65	0.29	-0.3	0.48	0.54	0.39
0.64	-0.3	0.16	0.56	-0.56	0.4

Tab. 15 Baum 8

13.4 Collage-Theorem

Wie können die bisher besprochenen, komplexen Strukturen erzeugt werden? Färbt man bei einem der beiden Bäume alle Punkte einer bestimmten affinen Abbildung mit gleicher Farbe, so fällt auf, daß das ganze Bild in Teilstrukturen einer Farbe zerfällt. Dies ist der Inhalt des sog. Collage-Theorems von Barnsley (1985). Soll eine komplexe Struktur mittels IFS erzeugen werden, so muß man verkleinerte Kopien des selbstähnlichen Gebildes so zusammensetzen, daß keine Lücken bleiben. Die Vereinigungsmenge der Bildpunkte ergibt dann – anschaulich gesprochen – den gesuchten Attraktor. Eine genauere Formulierung findet sich bei Barnsley [02].

Mit Hilfe des Collage-Theorems sollen zwei Blattstrukturen erzeugt werden. Besonders schön läßt sich das Collage-Theorem an dem folgenden Ahornblatt demonstrieren, wenn man die Einzelabbildungen in getrennten Farben ausdruckt.

a	b	c	d	e	f
0.65	0	0	0.6	0.175	0
0.65	0	0	0.65	0.165	0.325
0.32	-0.32	0.32	0.32	0.2	0
-0.32	0.32	0.32	0.32	0.8	0

Tab. 16 Blatt 1

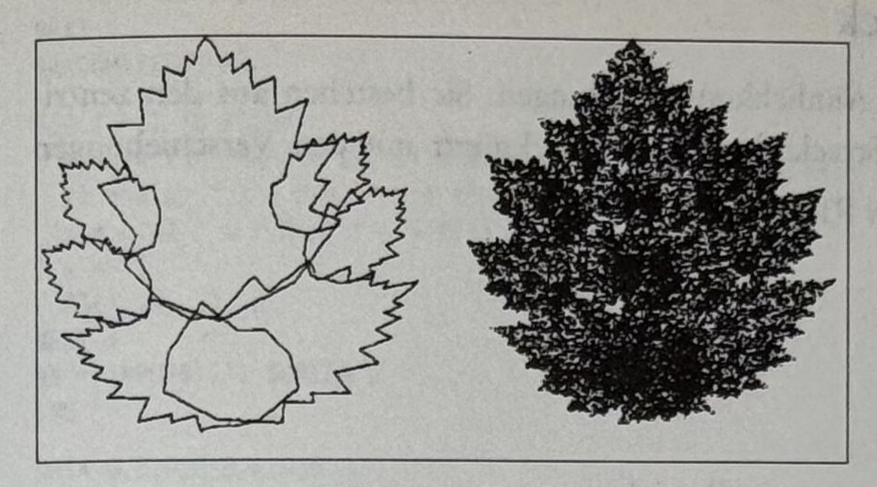


Bild 13.5 Collage eines Blattes

Ein Ahornblatt mit Stiel wird erzeugt durch die nachfolgende IFS

a	b	c	d	e	f
0.352	0.355	-0.355	0.352	0.354	0.5
0.353	-0.354	0.354	0.353	0.288	0.153
0.5	0	0	0.5	0.25	0.462
0.502	-0.002	0.002	0.588	0.25	0.105
0.004	0	0	0.578	0.501	0.06

Tab. 17 Blatt 2

Ein spitz zulaufendes Blatt erhält man durch die nachfolgenden Affinitäten

a	b	c	d	e	f
0.29	0.4	-0.4	0.3	0.28	0.44
0.33	-0.34	0.39	0.4	0.41	0
0.42	0	0	0.63	0.29	0.36
0.61	0	0	0.61	0.19	0.23
0.01	0	0	0.29	0.5	0.13

Tab. 18 Blatt 3

13.5 Sierpinski-Dreieck

Das Sierpinski-Dreieck hat 3 Ähnlichkeitsabbildungen. Sie bestehen aus den zentrischen Streckungen mit dem Streckfaktor $k=\frac{1}{2}$, verknüpft mit den Verschiebungen um die Strecke $\frac{1}{2}$ in x- bzw. y Richtung.

$$w_1(x,y)=(\tfrac{1}{2}x,\tfrac{1}{2}y)$$
$$w_2(x,y)=(\tfrac{1}{2}x+\tfrac{1}{2},\tfrac{1}{2}y)$$
$$w_3(x,y)=(\tfrac{1}{2}x,\tfrac{1}{2}y+\tfrac{1}{2})$$

Schreibt man dies in Matrixform, so ergibt sich

$$w_1:\begin{pmatrix}x\\y\end{pmatrix}\rightarrow\begin{pmatrix}\frac{1}{2}&0\\0&\frac{1}{2}\end{pmatrix}+\begin{pmatrix}0\\0\end{pmatrix}$$

und analog

$$w_2:\begin{pmatrix}x\\y\end{pmatrix}\rightarrow\begin{pmatrix}\frac{1}{2}&0\\0&\frac{1}{2}\end{pmatrix}+\begin{pmatrix}\frac{1}{2}\\0\end{pmatrix};\quad w_3:\begin{pmatrix}x\\y\end{pmatrix}\rightarrow\begin{pmatrix}\frac{1}{2}&0\\0&\frac{1}{2}\end{pmatrix}+\begin{pmatrix}0\\\frac{1}{2}\end{pmatrix}$$

In Form einer Tabelle liefert dies den IFS-Code

a	b	c	d	e	f
0.5	0	0	0.5	0	0
0.5	0	0	0.5	0.5	0
0.5	0	0	0.5	0	0.5

Tab. 19 IFS-Code für Sierpinski-Dreieck

Als Beispiel eines Basic-Programms soll hier ein Programm `sierp2.bas` zur Erzeugung des Sierpinski-Dreiecks gegeben werden. Der IFS-Code wird hier in Form von DATA-Werten eingelesen.

```
'sierp2.bas
'Sierpinski-Kurve mittels IFS

DIM i AS LONG
DIM p AS INTEGER
DIM x, x1, y AS SINGLE
DIM a(3), b(3), c(3), d(3), e(3), f(3)

SCREEN 12: CLS
WINDOW (-.05, -.05)-(1.05, 1.05)
FOR i = 1 TO 3
```

```
  READ a(i), b(i), c(i), d(i), e(i), f(i)
NEXT i
RANDOMIZE TIMER

FOR i = 1 TO 75000
  p = INT(3 * RND) + 1: 'alle Abbildungen gleichwahrscheinlich
  x1 = a(p) * x + b(p) * y + e(p)
  y = c(p) * x + d(p) * y + f(p)
  x = x1
  PSET (x, y), p
NEXT i
e$ = INPUT$(1): SCREEN 0
END

DATA 0.5,0,0,0.5,0,0
DATA 0.5,0,0,0.5,0.5,0
DATA 0.5,0,0,0.5,0,0.5
```

13.6 Koch-Kurve

Die Koch-Kurve hat 4 Ähnlichkeitsabbildungen. Zwei dieser Abbildungen sind zentrische Streckungen mit dem Streckfaktor $k=\frac{1}{3}$; eine davon ist mit einer Verschiebung um die Strecke $\frac{2}{3}$ in x-Richtung verknüpft.

Die anderen beiden sind Drehstreckungen mit dem Drehwinkel ±60°, verknüpft mit den Verschiebungen$\left(\frac{1}{3},0\right)$ bzw. $\left(\frac{2}{3},0\right)$. Die Abbildungen lauten somit

$$w_1(x,y)=\left(\frac{1}{3}x,\frac{1}{3}y\right)$$
$$w_2(x,y)=\left(\frac{1}{3}x+\frac{2}{3},\frac{1}{3}y\right)$$
$$w_3(x,y)=(\frac{1}{6}x-\frac{1}{6}\sqrt{3}y+\frac{1}{3},\frac{1}{6}\sqrt{3}x+\frac{1}{6}y)$$
$$w_4(x,y)=(-\frac{1}{6}x+\frac{1}{6}\sqrt{3}y+\frac{2}{3},\frac{1}{6}\sqrt{3}x+\frac{1}{6}y)$$

Schreibt man dies als Matrix-Abbildungen, so ergibt sich

$$w_1\colon\begin{pmatrix}x\\y\end{pmatrix}\to\begin{pmatrix}\frac{1}{3}&0\\0&\frac{1}{3}\end{pmatrix}\begin{pmatrix}x\\y\end{pmatrix};\quad w_2\colon\begin{pmatrix}x\\y\end{pmatrix}\to\begin{pmatrix}\frac{1}{3}&0\\0&\frac{1}{3}\end{pmatrix}\begin{pmatrix}x\\y\end{pmatrix}+\begin{pmatrix}\frac{2}{3}\\0\end{pmatrix}$$

und

$$w_3: \begin{pmatrix} x \\ y \end{pmatrix} \to \begin{pmatrix} \frac{1}{6} & -\frac{1}{6}\sqrt{3} \\ \frac{1}{6}\sqrt{3} & \frac{1}{6} \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix} + \begin{pmatrix} \frac{1}{3} \\ 0 \end{pmatrix}; \quad w_4: \begin{pmatrix} x \\ y \end{pmatrix} \to \begin{pmatrix} \frac{1}{6} & \frac{1}{6}\sqrt{3} \\ \frac{1}{6}\sqrt{3} & \frac{1}{6} \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix} + \begin{pmatrix} \frac{2}{3} \\ 0 \end{pmatrix}$$

Daraus ergibt sich folgender IFS-Code

a	b	c	d	e	f
0.3333	0	0	0.3333	0	0
0.3333	0	0	0.3333	0.6667	0
0.1667	-0.28867	0.28867	0.1667	0.3333	0
-0.1667	0.28867	0.28867	0.1667	0.667	0

Tab. 20 IFS-Code für Koch-Kurve

Als Beispiel eines C-Programms für ein IFS wird das Programm `kochifs.c` gegeben.

```
/* kochifs.c */
/* IFS für Koch-Kurve */
#include <stdio.h>
#include <graphics.h>
#include <conio.h>
#include <stdlib.h>
#include <time.h>

float a[4] = {0.3333,0.3333,0.1667,-0.1667};
float b[4] = {0,0,-0.28867,0.28867};
float c[4] = {0,0,0.28867,0.28867};
float d[4] = {0.3333,0.3333,0.1667,0.1667};
float e[4] = {0,0.6667,0.3333,0.6667};
float f[4] = {0,0,0,0};

void main(void)
{
long int i;
int k;
float x,y,x1;
time_t now;
int gdriver,gmode;
gdriver = 9; gmode = VGAHI; /* VGA-Modus 640 x 480 */
initgraph(&gdriver,&gmode," ");
srand((unsigned) time(&now) % 4001);

x = y = 0;
for (i=1; i<=80000L; i++)
  {
  k = rand() % 4; /* alle Abbildungen gleichwahrscheinlich */
  x1 = a[k] * x + b[k] * y + e[k];
  y = c[k] * x + d[k] * y + f[k];
  x = x1;
  putpixel((int)(600*x)+20,300-(int)(600*y),k+1);
```

```
  }
do {} while(!(kbhit()));
closegraph();
return;
}
```

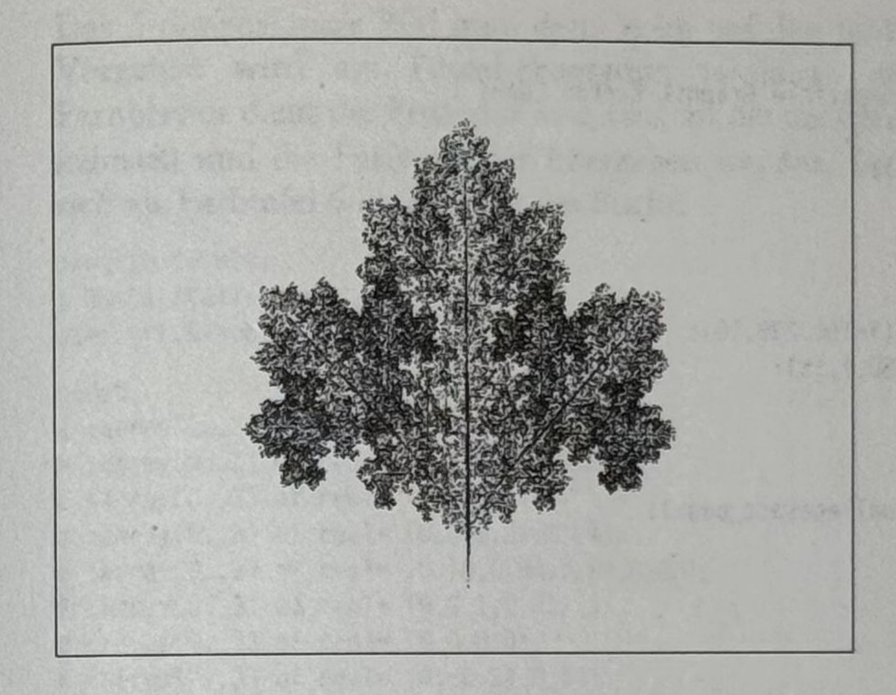


Bild 13.6 IFS nach Tab 18.

13.7 Deterministisches IFS-Programm

Das von Barnsley in *Fractals Everywhere* angegebene deterministische IFS-Programm für das Sierpinski-Dreieck ist sehr speicherplatzaufwendig, da alle Punkte in 2 zweidimensionalen Reihungen (array) gespeichert werden müssen. Dies führt dazu, daß bei einem 64K-Speicher pro Variable nur mit einem Gitter von 100x100 Pixeln gearbeitet werden kann. Eleganter ist das Vorgehen, die gesetzten Pixel direkt vom Bildschirm zu lesen und das Umspeichern der Pixel durch Umschalten der aktuellen Bildschirmseite zu bewerkstelligen.

Das Turbo Pascal-Programm `ifsdet.pas` arbeitet auf einem 300x300-Gitter und könnte daher auch im EGA-Modus betrieben werden. Dies wird auch empfohlen, wenn die verwendete Videoplatine im VGA-Modus nicht vollkommen kompatibel ist.

```
program ifsdet;
{ Deterministischer Algorithmus für Sierpinski-IFS }
uses crt,graph;

const
a: array[0..3] of real = (0.5 ,0.5, 0.5,0);
b: array[0..3] of real = (0,0,0,0);
c: array[0..3] of real = (0,0,0,0);
d: array[0..3] of real = (0.5,0.5,0.5,0);
e: array[0..3] of real = (75,0,150,0);
```

```
f: array[0..3] of real = (0,150,150,0);
act_page: integer = 1;
last_page: integer = 0;
var i,j,k,m,n,t,iter,color: integer;
    GraphDriver,GraphMode : integer;

begin
GraphDriver := 4;
GraphMode := VGAHi; { Für nichtkompatible Graphik-Karten EGAhi }
InitGraph(graphDriver,GraphMode,'');
SetVisualPage(0);SetActivePage(0);
ClearDevice;

for i:=0 to 299 do
  begin
  PutPixel(i+150,0,15); PutPixel(i+150,299,15);
  PutPixel(449,i,15); PutPixel(150,i,15);
  end;
for iter :=0 to 8 do
  begin
  SetActivePage(act_page);SetVisualPage(act_page);
  ClearDevice;
  for i:=0 to 300 do
  for j:=0 to 300 do
    begin
    SetActivePage(last_page);
    color := getPixel(i+150,j);
    if color <> 0 then
    for k:=0 to 2 do
      begin
      n := Round(a[k]*i + b[k]*j + e[k]);
      m := Round(c[k]*i + d[k]*j + f[k]);
      SetActivePage(act_page);
      PutPixel(n+150,m,15);
      end;
    end;
   t:= act_page; act_page := last_page;last_page := t;
end;
repeat until keypressed;
closegraph;
textmode(lastmode)
end.
```

13.8 3D-Darstellung eines IFS

IFS-Abbildungen können auch auf drei Variable verallgemeinert werden. Man erhält dann Abbildungen des Raumes auf sich

$$\begin{pmatrix} x \\ y \\ z \end{pmatrix} \rightarrow \begin{pmatrix} a & b & c \\ d & e & f \\ g & h & m \end{pmatrix} \begin{pmatrix} x \\ y \\ z \end{pmatrix} + \begin{pmatrix} n \\ q \\ r \end{pmatrix}$$

Das 3-dimensionale Bild muß dann noch auf den Bildschirm projiziert werden. Das Vorgehen wird am Pascal-Programm `farn3d.pas` erklärt. Zum Zeichnen eines Farnblattes dient die Prozedur `make_farn`, an die die drei Drehwinkel α, β, γ (im Winkelmaß) und die Farbnummer übergeben werden. Die resultierende Graphik findet sich als Farbtafel 6 im Farbteil des Buchs.

```
program farn3d;
{ 3D-Darstellung des Farnkrauts }
uses crt,Graph;

const
a :array[0..3] of real= (0,0.83,0.22,-0.22);
b :array[0..3] of real= (0,0,-0.23,0.23);
c :array[0..3] of real= (0,0,0,0);
d :array[0..3] of real= (0,0,0.24,0.24);
e :array[0..3] of real= (0.18,0.86,0.22,0.22);
f :array[0..3] of real= (0,0.1,0,0);
g :array[0..3] of real= (0,0,0,0);
h :array[0..3] of real= (0,-0.12,0,0);
m :array[0..3] of real= (0,0.84,0.32,0.32);
n :array[0..3] of real= (0,0,0,0);
q :array[0..3] of real= (0,1.62,0.82,0.82);
r :array[0..3] of real= (0,0,0,0);
p :array[0..3] of real= (0.01,0.85,0.92,1.0);
var GraphDriver,GraphMode: integer;

procedure make_fern(alpha,beta,gamma,color:integer);
var i,k : integer;
    ca,cb,cg,sa,sb,sg,vx,vy,x,y,z,x1,y1,pk: real;
begin
ca := cos(alpha*0.0174533); cb := cos(beta*0.0174533);
cg := cos(gamma*0.0174533); sa := sin(alpha*0.0174533);
sb := sin(beta*0.0174533);  sg := sin(gamma*0.0174533);
x := 0; y := 0; z := 0;
for i:=1 to 25000 do
  begin
  pk := random;
  if pk < p[0] then k := 0
    else if pk < p[1] then k := 1
    else if pk < p[2] then k := 2
    else k := 3;
  x1 := (a[k]* x + b[k] * y + c[k] * z + n[k]);
  y1 := (d[k]* x + e[k] * y + f[k] * z + q[k]);
  z := (g[k] * x + h[k] * y + m[k] * z + r[k]);
  x := x1; y := y1;
```

```
  vx := x*ca + y*cb + z*cg; { Projektion }
  vy := x*sa + y*sb + z*sg;
  PutPixel(round(vx*40)+365,400-round(vy*50),color);
end;

end;
begin
GraphDriver := 9;
GraphMode := VGAHi; { VGA-Modus }
InitGraph(graphDriver,GraphMode,'');
ClearDevice;
randomize;
make_fern(30,115,25,2);
make_fern(45,105,70,14);
make_fern(15,70,10,11);
make_fern(-20,70,120,10);
repeat until keypressed;
closegraph;
textmode(lastmode)
end.
```

13.9 Ausblicke und Ergänzungen

Übung 13.9.1

Realisieren Sie den Binärbaum, der durch folgenden IFS-Code gegeben ist.

a	b	c	d	e	f
0	0	0	0.5	0	0
0.4	-0.4	0.4	0.4	0	0.2
0.4	0.4	-0.4	0.4	0	0.2

Tab. 21 Binärbaum

Übung 13.9.2

Das Dürer-Fünfeck hat folgenden IFS-Code

a	b	c	d	e	f
0.382	0	0	0.382	0.3072	0.6190
0.382	0	0	0.382	0.6033	0.4044
0.382	0	0	0.382	0.0139	0.4044
0.382	0	0	0.382	0.1253	0.0595
0.382	0	0	0.382	0.4920	0.0595

Tab. 22 Dürer-Fünfeck

Übung 13.9.3

Einen Ternärbaum oder 3-strahligen Stern liefert der folgende IFS-Code

a	b	c	d	e	f
0.255	0	0	0.255	0.3726	0.6714
0.255	0	0	0.255	0.1146	0.2232
0.255	0	0	0.255	0.6306	0.2232
0.370	-0.642	0.642	0.370	0.6356	-0.0061

Tab. 23 Ternärbaum

Übung 13.9.4

Eine Spirale mittels Kreisinversion und Drehstreckung zeigt folgender IFS-Code

a	b	c	d	e	f
0.787879	-0.424242	0.242424	0.859848	1.758647	1.408065
-0.121212	0.257576	0.151515	0.053030	-6.721654	1.377236
0.181818	-0.136364	0.090909	0.181818	6.086107	1.568035

Tab. 24 Spirale

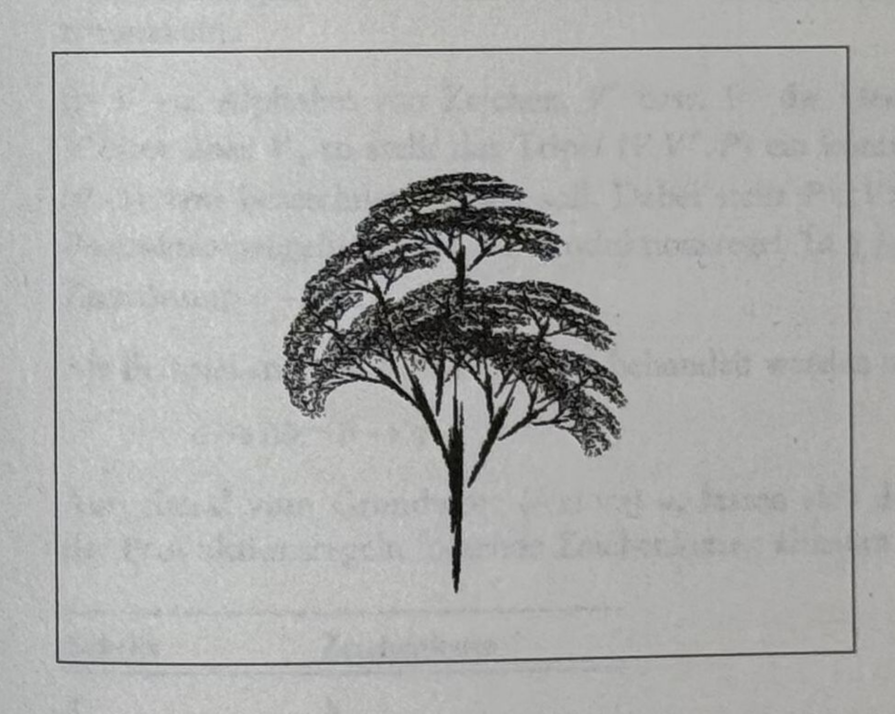


Bild 13.7 IFS nach Tab. 15

Übung 13.9.5

Ebenfalls eine Kreisabbildung ist folgender IFS-Code für Farbtafel 4

a	b	c	d	e	f
0.894130	0.102622	-0.102622	0.894130	0	0.1
0.894449	0.0998037	0.0998037	0.894449	0.1	0
0.0570061	0	0	0.0525302	0.9226135	0.492980
0.0565082	0	0	0.0499227	0.4754304	0.922063
0.0569784	0	0	0.0516372	0.0101721	0.481853
0.0564028	0	0	0.0504088	0.4433266	0.013917

Tab.25 Kreisabbildung



14 Lindenmayer-Systeme

14.1 Definition von L-Systemen

1956 publizierte N. Chomsky die Theorie der formalen Grammatiken. Neben Chomsky wurde auch 1959/60 von J. W. Backus und P. Naur eine formale Beschreibung von Programmiersprachen angegeben, die sie zur Definition von ALGOL60 entwickelt hatten, die nunmehr BNF (*Backus-Naur-Form)* genannt wird. Backus war übrigens maßgeblich beim Entwurf von FORTRAN beteiligt gewesen. Von S.Ginsburg u.a. wurde dann 1962 die Gleichwertigkeit der BNF-Beschreibung mit den kontextfreien Chomsky-Grammatiken erkannt. Auf Grund dieser formalen Beschreibungen konnte dann 1968 A. Lindenmayer die Theorie der L-Systeme entwickeln.

Ist V ein Alphabet von Zeichen, V^* bzw. V^+ die Menge aller bzw. aller nichtleerer Wörter über V, so stellt das Tripel (V, V^+, P) ein kontextfreies L-System dar, das als *0L-System* bezeichnet werden soll. Dabei stellt $P \subset V \times V^*$ die (endliche) Menge der *Produktionsregeln* dar. Eine Produktionsregel $(a, \chi) \in P$ kann man schreiben als Zuordnung $a \to \chi$.

Als Beispiel soll das Alphabet $\{a,b\}$ behandelt werden mit den Produktionsregeln

$$a \to ab; \quad b \to a$$

Ausgehend vom Grundwort (Axiom) a, lassen sich durch gleichzeitige Anwendung der Produktionsregeln folgende Zeichenketten ableiten

Schritt	Zeichenkette
1	b
2	a
3	ab
4	aba

Schritt	Zeichenkette
5	*abaab*
6	*abaababa*
7	*abaababaabaab*

Ein 0L-System ist somit charakterisiert durch die Angabe eines Axioms und der zugehörigen Produktionsregeln. Das 0L-System für die Kochsche Kurve besteht aus dem Axiom F und den Produktionsregeln

$$F \to F-F++F-F; \quad + \to +; \quad - \to -$$

Dabei kann F als *Forward*; d.h. als Zeichnen einer Strecke, + als Linksdrehung und – als Rechtsdrehung interpretiert werden. Die Gültigkeit der beiden letztgenannten Produktionsregeln wird, sofern nicht anderes angegeben ist, als Identitätsregeln stillschweigend vorausgesetzt.

14.2 Die Turtle-Interpretation

P. Prusinkiewcz konnte 1986 eine Interpretation der 0L-Systeme mit Hilfe der von S. Papert 1980 entwickelten Turtle-Graphik angeben.

Symbol	Turtle-Aktion
F	Vorwärtsbewegung mit Zeichenstift
f	Vorwärtsbewegung ohne Zeichenstift
+	Linksdrehung um vorgebenen Winkel
-	Rechtdrehung um vorgebenen Winkel
[	Speichern des Turtle-Zustands auf einem Stack
]	Abruf des Turtle-Zustands vom Stack

Sind (x,y) die momentanen Koordinaten der Turtle und ist ihre Richtung (Heading) durch den Winkel φ gegeben, so kann der Turtle-Zustand durch das Tripel (x,y,φ) beschrieben werden. Durch die Kommandos ändert sich der Turtle-Zustand beim vorgebenen Drehwinkel δ wie folgt

Kommando	neuer Zustand
F	$(x+l\cos\varphi, y+l\sin\varphi, \varphi)$
+	$(x,y,\varphi-\delta)$
-	$(x,y,\varphi+\delta)$

Diese Rechenoperationen können leicht in einer Programmiersprache realisiert werden.

14.3 Pflanze1

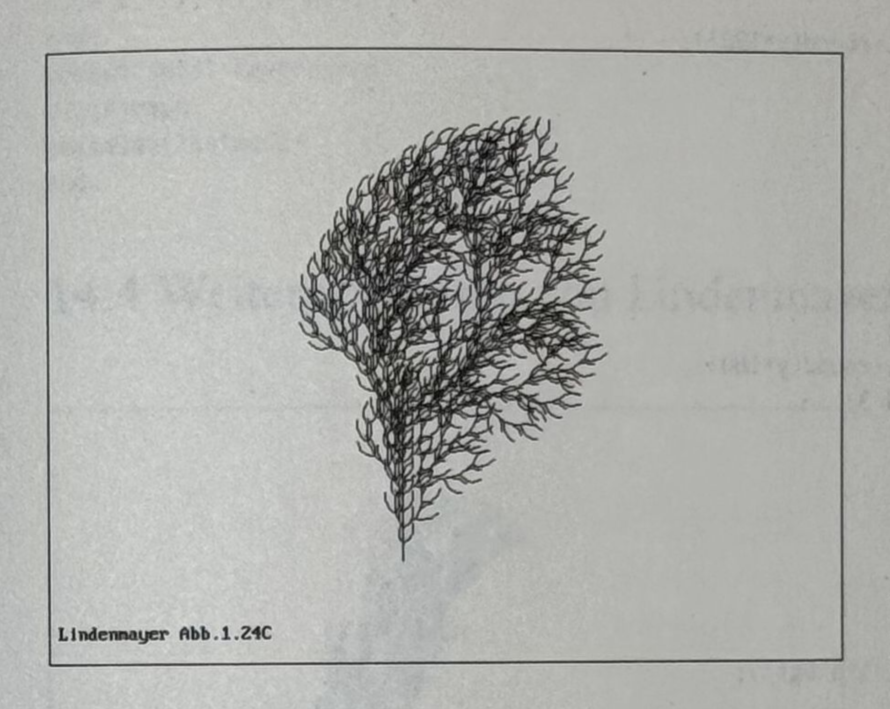


Bild 14.1 Busch nach Lindenmayer 1.24C

Als erstes Beispiel soll die Pflanze von Lindenmayers Buch [27] Seite 25, Bild 1.24c, simuliert werden. Sie hat das Axiom F, die Produktionsregel

$$F \rightarrow FF-[-F+F+F]+[+F-F-F]$$

Die Rekursionstiefe soll $n=4$, der Winkel $\delta = 22.5^\circ$ betragen.

Bild 14.1 bzw. Farbtafel 8 wird erzeugt von dem Turbo Pascal-Programm `busch.pas`.

```
program busch;
{ Lindenmayer, Algorithmic Beauty of Plants, Bild 1.24C }

uses crt,graph;

const r = 4;   { Rekursionsstufe}
var col,i,lp,m,nl,s : integer;
    delta,h,phi,x,y : real;
    u,v,w : array[1..16] of real;  { Stacks }
    n : array[1..250] of integer;
    p,prod : string;
    axiom,q,t : char;
    Graphdriver,Graphmode:integer;

procedure graphics;
```

```
begin
moveto(round(x*200)+280,-round(y*180));
case t of
   '+': phi := phi + delta;
   '-': phi := phi - delta;
   'F': begin
        x := x + h * cos(phi);
        y := y + h * sin(phi);
        lineto(round(x*200)+280,-round(y*180));
        end;
   '[': begin
        m := m + 1; u[m] := x;
        v[m] := y; w[m] := phi;
        end;
   ']': begin
        x := u[m]; y := v[m];
        phi := w[m]; m := m - 1;
        moveto(round(x*200)+280,-round(y*180));
        col := 1 + (1 + col) mod 3;
        setcolor(col)
        end
   end
end;

begin
GraphDriver := Detect;
Initgraph(GraphDriver,GraphMode,'\tp\bgi');
SetgraphMode(Graphmode);

axiom := 'F';                         { Axiom }
prod := 'FF-[-F+F+F]+[+F-F-F]';       { Produktionsregel }
x := 0; y := -2.5;                    { Startposition }
phi := pi/2;                          { Orientierung }
delta := pi/7; h := 0.04;             { Winkel,Schrittweite }
lp := length(prod);

s := 1; nl := 0; p[1] := 'F';
n[1] := 0; m := 0;
col := 1; setcolor(col);
while s > 0 do
begin
   while nl < r do
   begin
      t := p[s]; nl := n[s] + 1; s := s - 1;
      case t of
'F': for i := 0 to lp - 1 do
         begin
            q := prod[lp-i];
            s := s + 1;
            p[s] := q; n[s] := nl
         end;
      '+','-','[',']':
```

```
          begin
            s := s + 1; p[s] := t; n[s] := n1
          end;
      end;
   end;
t := p[s];
graphics;
s := s - 1; n1 := n[s]
end;
repeat until keypressed;
closegraph;
textmode(lastmode)
end.
```

14.4 Weitere Pflanzen von Lindenmayer

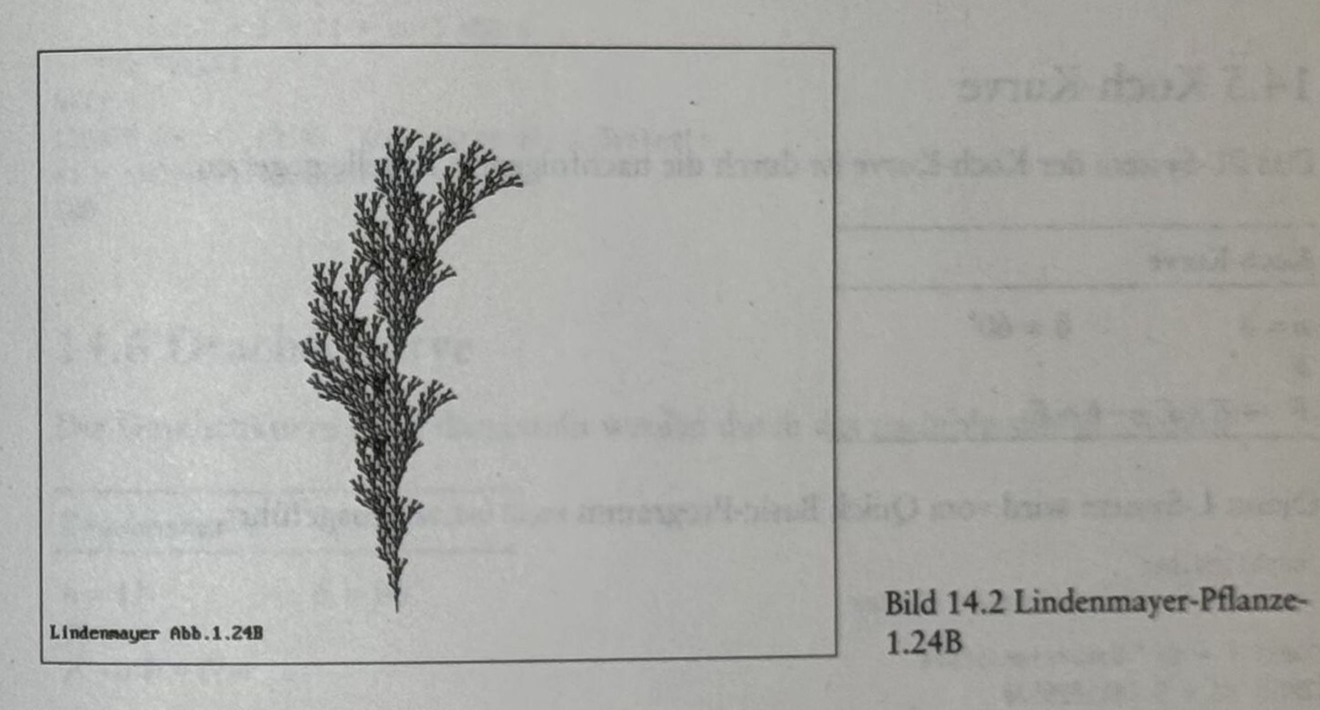


Bild 14.2 Lindenmayer-Pflanze-1.24B

Die Pflanze der Lindenmayer-Abb. 1.24A hat das Axiom F, die Produktionsregel

$$F \to F[+F]F[-F][F]$$

Die Rekursionstiefe soll $n = 4$, der Winkel $\delta = 20^\circ$ betragen. Es ergibt sich das Bild 14.2. Ähnlich ist die Pflanze der Lindenmayer-Abbildung 1.24B. Sie hat dieselbe Produktionsregel. Jedoch beträgt die Rekursionstiefe $n = 5$, der Winkel $\delta = 25.7^\circ$ (Bild 14.3.)

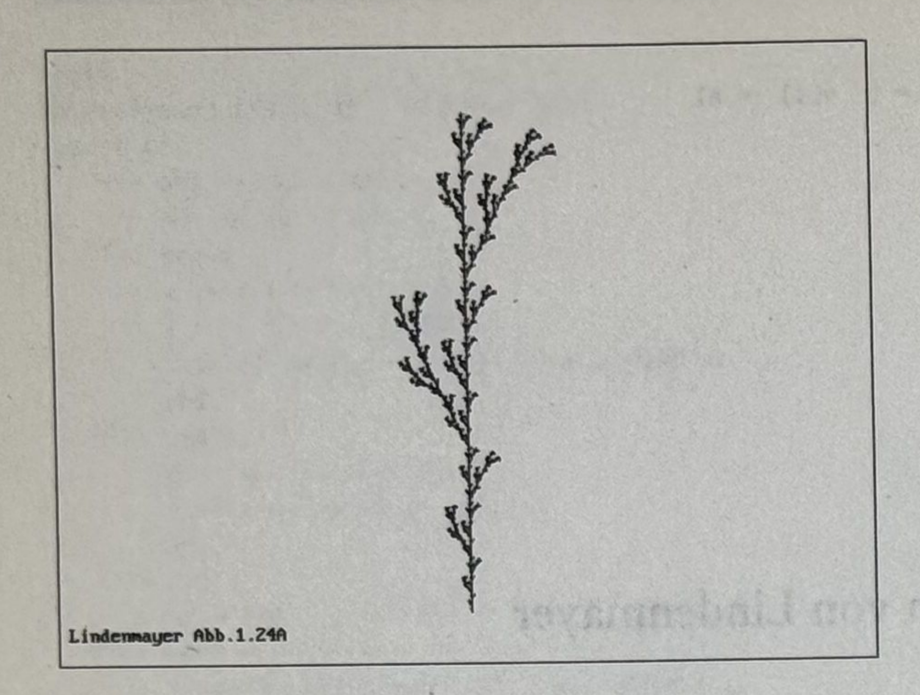


Bild 14.3 Lindenmayer-Pflanze 1.24A

14.5 Koch-Kurve

Das 0L-System der Koch-Kurve ist durch die nachfolgende Tabelle gegeben.

Koch-Kurve	
$n=4$	$\delta=60^\circ$
F	
$F \rightarrow F+F--F+F$	

Dieses L-System wird vom Quick Basic-Programm `kochlind.bas` ausgeführt.

```
'kochlind.bas
'Koch-Kurve als Lindenmayer-System

CONST r = 5: 'Rekursionsstufe
CONST PI = 3.141529653#
DIM i, j AS INTEGER
DIM h, phi, x, y AS DOUBLE

SCREEN 12: CLS
WINDOW (-4, -3)-(4, 3)
LINE (-4, -3)-(4, 3), , B

x = -3.6: y = -.5: phi = 0: h = .03: col = 1
AXIOM$ = "F"
PROD$ = "F+F--F+F"
way$ = AXIOM$
FOR j = 1 TO r
  w$ = ""
  FOR i = 1 TO LEN(way$)
```

```
      s$ = MID$(way$, i, 1)
      IF s$ = "F" THEN q$ = PROD$ ELSE q$ = s$
      w$ = w$ + q$
   NEXT i
   way$ = w$
NEXT j
PRESET (x, y), col
FOR i = 1 TO LEN(way$)
   s$ = MID$(way$, i, 1)
   SELECT CASE s$
   CASE "+"
        phi = phi + PI / 3
   CASE "-"
        phi = phi - PI / 3
   CASE "F"
        x = x + h * COS(phi)
        y = y + h * SIN(phi)
        LINE -(x, y), col
        col = 1 + (1 + col) MOD 5
   END SELECT
NEXT i
LOCATE 29, 4: PRINT "Koch-Kurve als L-System";
e$ = INPUT$(1): SCREEN 0
END
```

14.6 Drachenkurve

Die Drachenkurve kann dargestellt werden durch das nachfolgende 0L-System

Drachenkurve	
$n=11$	$\delta=90^\circ$
F	
$F \to F+G+$	
$G \to -F-G$	

Da diese Kurve zwei Produktionsregeln hat, muß das Basic-Programm geeignet modifiziert werden.

```
'draglind.bas
'Drachenkurve mittels L-System

CONST r = 11: 'Rekursionstiefe
CONST PI = 3.141529653#
DIM col, i, j AS INTEGER
DIM delta, h, phi, x, y AS DOUBLE

SCREEN 12: CLS
```

```
WINDOW (-4, -3)-(4, 3)
LINE (-4, -3)-(4, 3), , B
AXIOM$ = "F"
PROD1$ = "F+G+"
PROD2$ = "-F-G"

x = 1: y = -1.6: 'Startposition
phi = 0: 'Orientierung
delta = PI / 2: h = .1: 'Winkel, Schrittweite
col = 1: way$ = AXIOM$
FOR j = 1 TO r
  w$ = ""
  FOR i = 1 TO LEN(way$)
    s$ = MID$(way$, i, 1)
    SELECT CASE s$
      CASE "F"
        q$ = PROD1$
      CASE "G"
        q$ = PROD2$
      CASE "+", "-"
        q$ = s$
      END SELECT
    w$ = w$ + q$
  NEXT i
  way$ = w$
NEXT j
PSET (x, y), 14
FOR i = 1 TO LEN(way$)
  s$ = MID$(way$, i, 1)
  SELECT CASE s$
    CASE "-"
      phi = phi - delta
    CASE "+"
      phi = phi + delta
    CASE "F", "G"
      x = x + h * COS(phi): y = y + h * SIN(phi)
      LINE -(x, y), col
      col = 1 + (1 + col) MOD 4
    END SELECT
NEXT i
e$ = INPUT$(1): SCREEN 0
END
```

14.7 Pfeilspitzkurve

Die Pfeilspitzkurve von Sierpinski ist gekennzeichnet durch das folgende 0L-System

Pfeilspitzkurve	
$n = 6$	$\delta = 60°$
G	
$F \rightarrow G + F + G$	
$G \rightarrow F - G - F$	

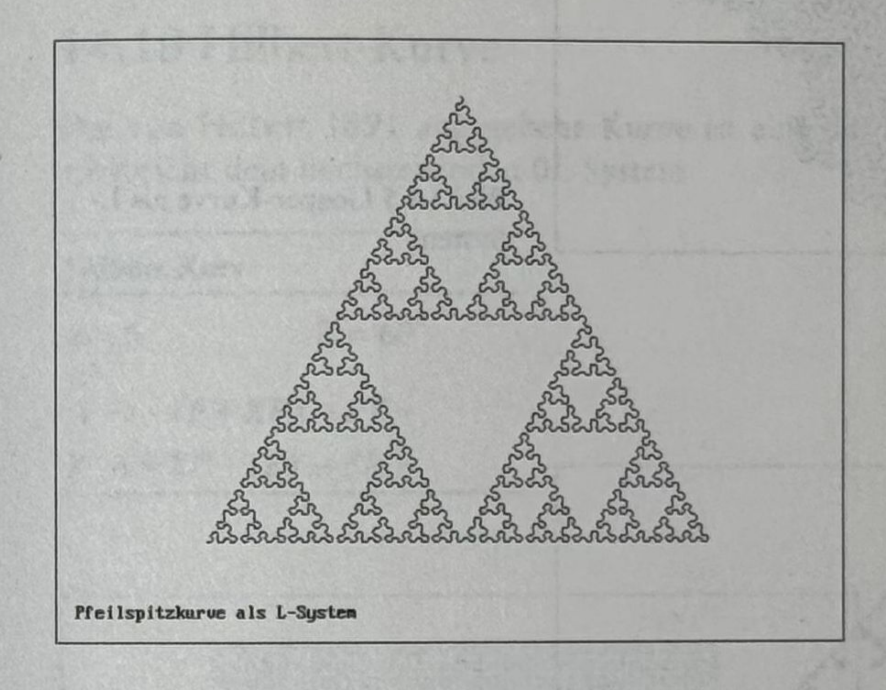


Bild 14.4 Pfeilspitzkurve mittels L-System

Die Kurve kann mit dem Programm zur Drachenkurve nach Ändern der Produktionsregeln erzeugt werden.

14.8 Hexagonale Gosper-Kurve

Die hexagonale Gosper-Kurve hat das folgende 0L-System

hexagonale Gosper-Kurve	
$n = 4$	$\delta = 60°$
F	
$F \rightarrow F + G + + G - F - - FF - G +$	
$G \rightarrow - F + GG + + G + F - - F - G$	

Die Kurve kann wieder mit dem Programm zur Drachenkurve realisiert werden.

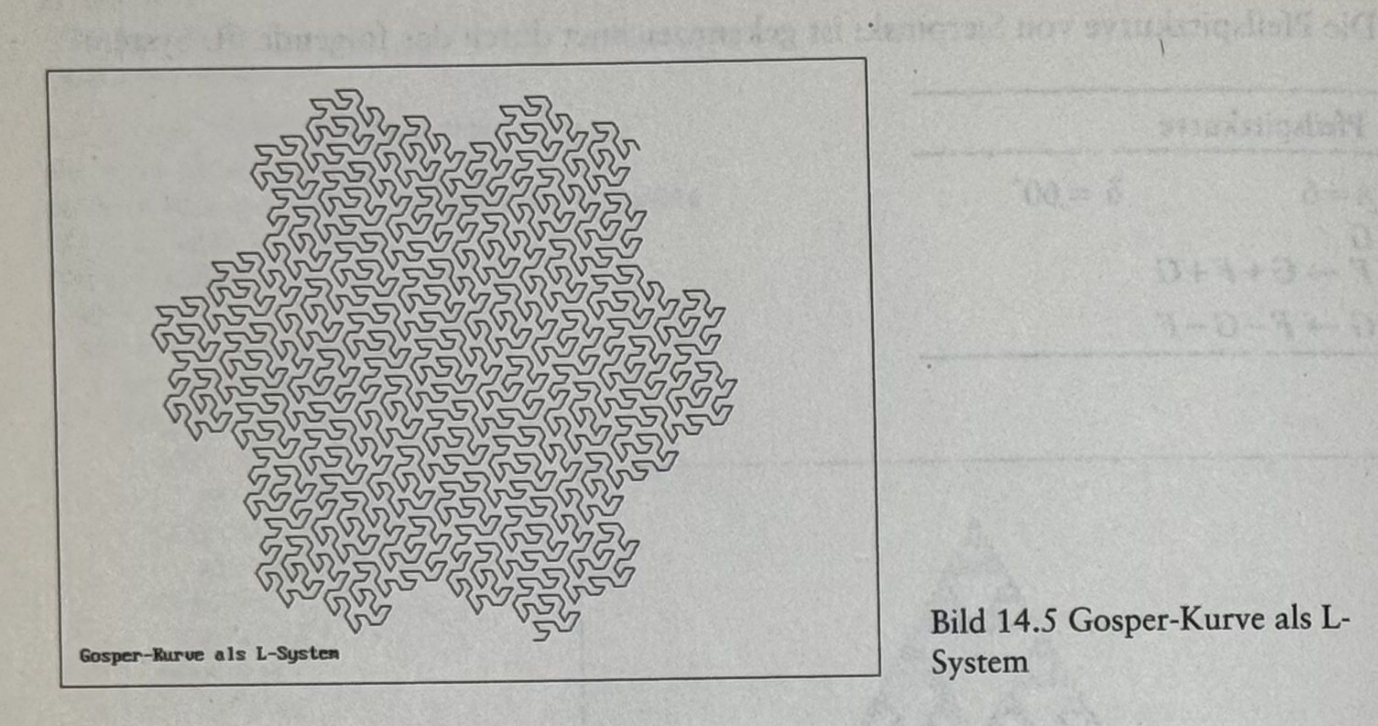


Bild 14.5 Gosper-Kurve als L-System

14.9 Kreuzstichkurve

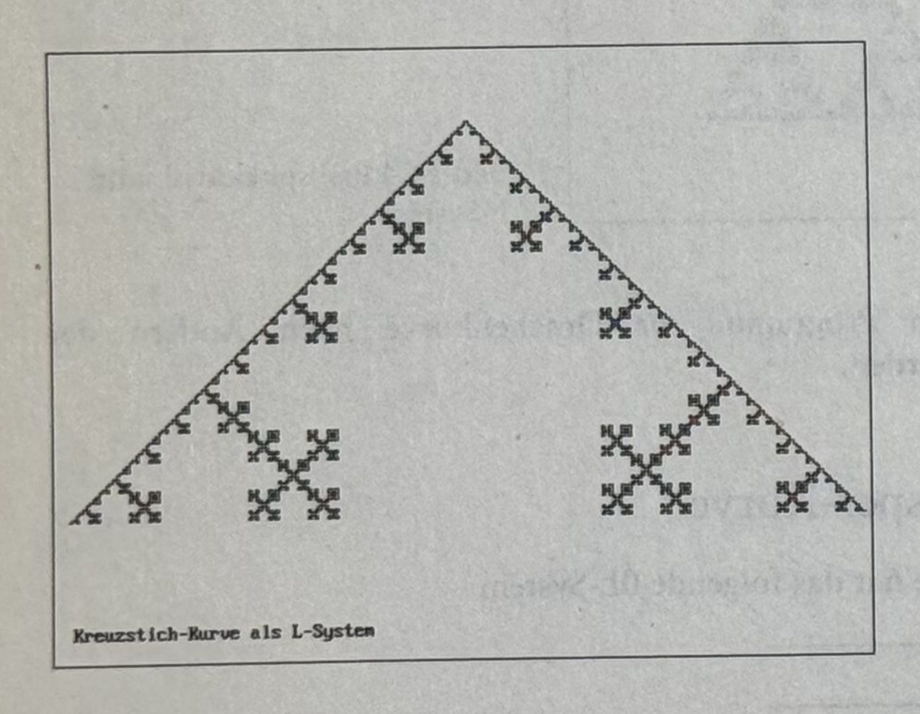


Bild 14.6 Kreuzstichkurve als L-System

Die Kreuzstichkurve von T. Vicsek hat das nachfolgende 0L-System

Kreuzstichkurve

$n = 4 \qquad \delta = 60°$
F
$F \rightarrow F + F - F - F + F$

Diese Kurve benötigt nur eine Produktionsregel; es kann daher das Programm `koch.bas` oder `busch.pas` verwendet werden.

14.10 Hilbert-Kurve

Die von Hilbert 1891 angegebene Kurve ist eine der bekannten Monsterkurven. Sie entspricht dem nachstehenden 0L-System

Hilbert-Kurve

$n = 6 \qquad \delta = 60°$
X
$X \rightarrow -YF + XFX + FY-$
$Y \rightarrow +XF - YFY - FX+$

Hilbert-Kurve als L-System

Bild 14.7 Hilbert-Kurve als L-System

Es kann wieder das Programm `draglind.bas` verwendet werden. Zum Start ist hier eine Drehung notwendig (Bild 14.7).

14.11 Minkowski-Kurve

Die Minkowski-Kurve hat das folgende 0L-System

Minkowski-Kurve	
$n = 3$	$\delta = 90°$
F	
$F \rightarrow F+F-F-FF+F+F-F$	

Da nur eine Produktionsregel benötigt wird, kann hier das Programm zur Koch-Kurve verwendet werden (vgl. Bild 14.8).

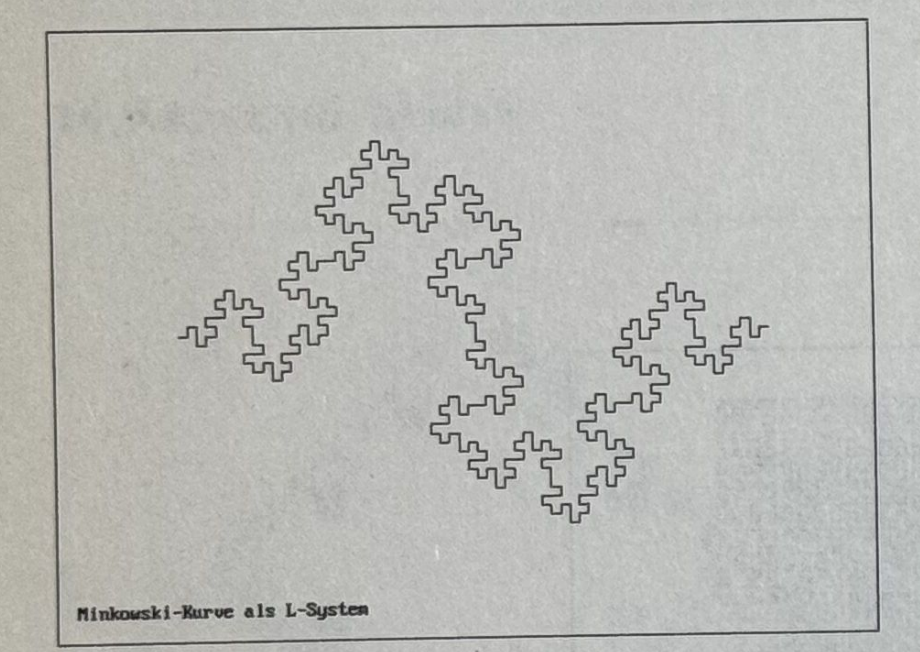


Bild 14.8 Minkowski-Kurve als L-System



15 Julia-Mengen

Julia-Mengen sind nach dem französischen Mathematiker Gaston Julia benannt, der 1918 mit seinem Aufsatz *Mémoire sur l'iteration des fonctions rationelles* im Journal de Math. Pure et Appl. seine Theorie publizierte. Die (gefüllte) Julia-Menge K_c der quadratischen komplexen Funktion $Q_c(z) = z^2 + c$ ist die Menge aller Punkte der komplexen Ebene, für die der Grenzwert der iterierten Abbildung beschränkt bleibt.

$$K_c = \left\{ z \in C;\ \lim_{n \to \infty} \left| Q_c^n(z) \right| < \infty \right\}$$

Die Julia-Menge J_c ist dann der Rand der Menge K_c: $J_c = \partial K_c$.Die ersten drei Iterierten der Funktion $Q_c(z) = z^2 + c$ sind dabei

$$z \to z^2 + c \to (z^2 + c)^2 + c \to ((z^2 + c)^2 + c)^2 + c$$

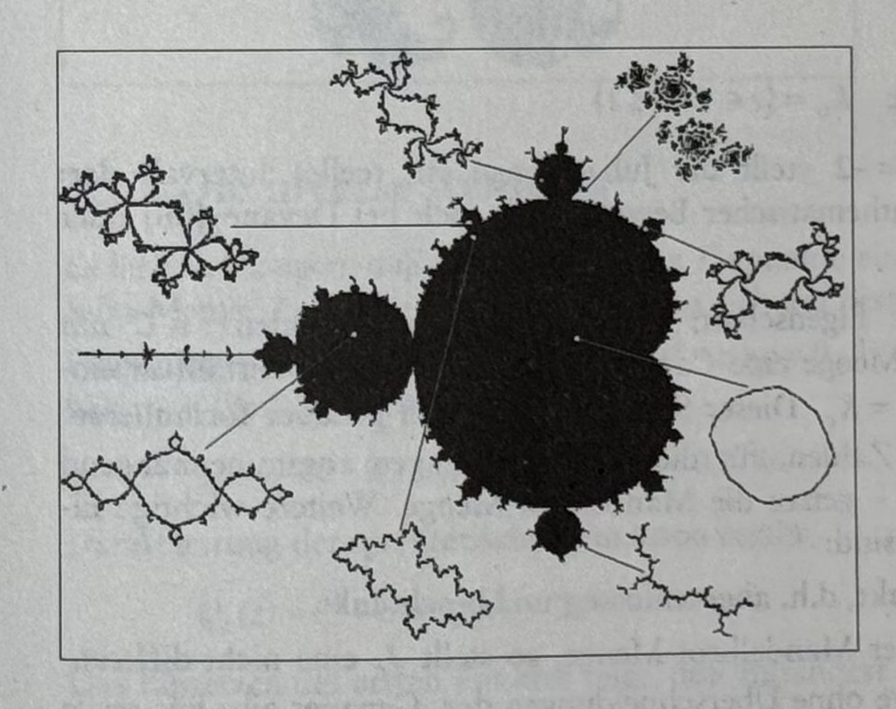


Bild 15.1 Lage der Parameter der Julia-Mengen am Apfelmännchen

Einige bekannte Julia-Mengen werden hier tabellarisch angeführt.

Parameter	Name	Abbildung
-1	San Marco	15.4
$0.360284+0.1i$	Julia-Drachen	15.2
$-0.122+0.745i$	Douadys Kaninchen	15.3
i	Dentrit	15.6
$0.11301-0.67037i$	Fatou-Staub	15.7
$-0.1011+0.9563i$	Misiurewicz-Punkt	15.5

Die zugehörigen Abbildungen finden sich verstreut in den folgenden Abschnitten. Sie können mit den angegebenen Programmen erzeugt werden, indem man die Parameter der Juliamenge, getrennt in Real- und Imaginärteil, als Konstante ins Programm eingibt; z.B. Douadys Kaninchen für $a=-0.122; b=0.745$. Die Lage der Parameter in der komplexen Zahlenebene für einige Julia-Mengen sind in Bild 15.1 ersichtlich. Die interessanten Julia-Mengen befinden sich am Rand des Apfelmännchens.

15.1 Eigenschaften von Julia-Mengen

Juliamengen sind für alle komplexen Zahlen außer für $c=0 \wedge c=-2$ fraktale Gebilde. Der Ausnahmefall $c=0$ ist leicht einzusehen. Für die Funktion $Q_o(z)=z^2$ ist nämlich der Einheitskreis invariant; d.h. Fixpunktmenge. Dies sieht man an der Betragsgleichung

$$|z|=1 \Rightarrow |z|^2=1$$

Es gilt somit $J_0=\{z \in C : |z|=1\}$; $K_0=\{z \in C : |z| \leq 1\}$

Im zweiten Ausnahmefall $c=-2$ stellt die Julia-Menge ein reelles Intervall dar: $J_{-2}=K_{-2}=[-2;2] \subset R$. Ein mathematischer Beweis findet sich bei Devaney[06] oder Zeitler[40].

Julia-Mengen zeigen folgende Eigenschaft: Für alle komplexen Zahlen $c \in C$ mit $|c|>2$ stellt die gefüllte Julia-Menge eine Cantor-Menge dar; d.h. sie zerfällt in einzelne Punkte. Es gilt dann $J_c=K_c$. Dieser Satz läßt sich noch genauer formulieren: Die Menge aller komplexen Zahlen, für die die Julia-Mengen zusammenhängend sind, ist – Sie ahnen es schon – genau die Mandelbrot-Menge. Weitere wichtige Eigenschaften der Julia-Mengen sind:

- Jede Julia-Menge ist kompakt, d.h. abgeschlossen und beschränkt.
- Gehört c zum »Bauch« der Mandelbrot-Menge, so stellt J_c eine nicht-differenzierbare, geschlossene Kurve ohne Überschneidungen dar. Genauer gilt: Für reelle $c \in \left[-\frac{3}{4};\frac{1}{4}\right]$ stellt J_c eine Jordankurve dar.

- Das Bild von J_c ist punktsymmetrisch.
- Gehört ein Punkt zu J_c, gehört auch jeder Vorgänger und Nachfolger des Punktes auf der Bahn von $Q_c(z)$ zu J_c.
- J_c ist ein Attraktor für die inverse Funktion $z \to Q_c(z)$. Gehört ein Punkt zu J_c, so wird J_c durch die Vorgänger des Punktes auf der Bahn von $Q_c(z)$ vollständig überdeckt.

Die drei letztgenannten Eigenschaften werden im folgenden Abschnitt zur Formulierung eines Algorithmus benutzt.

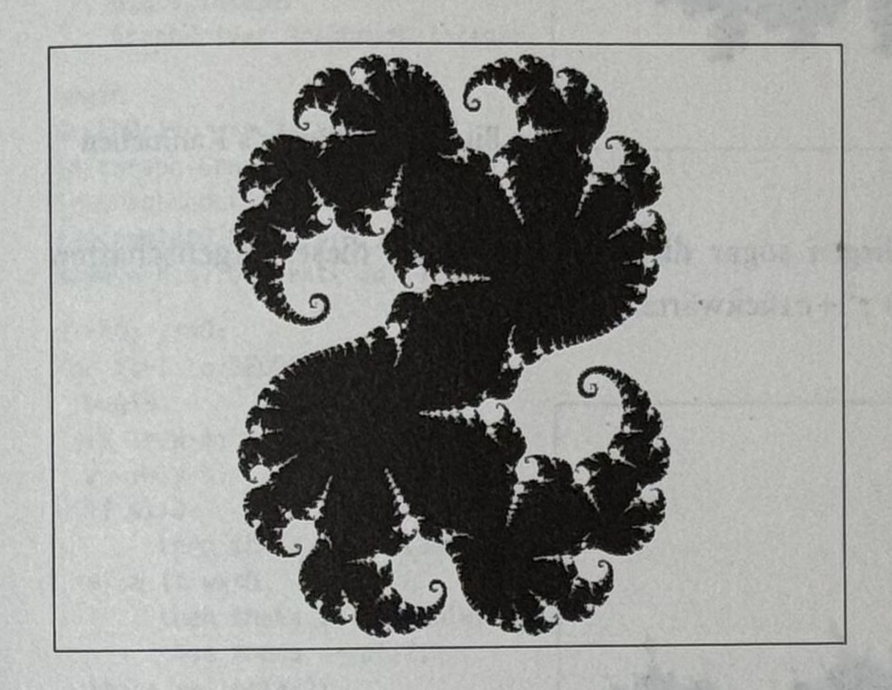


Bild 15.2 Julia-Drachen

15.2 Die inverse Iteration

Es läßt sich zeigen, daß alle abstoßenden Fixpunkte einer periodischen Bahn stets zur Julia-Menge J_c gehören. Dies macht man sich am besten an einem Beispiel klar. Die Fixpunktgleichung lautet $z = z^2 + c \Rightarrow z^2 - z + c = 0$. Für das Beispiel $c = 0.5 + 0.5i$ erhält man die Lösungen

$$z_1 = 1.408 - 0.275i; \quad z_2 = -0.408 + 0.275i.$$

Die Ableitung der quadratischen Funktion ergibt

$$Q_c(z) = z^2 + c \Rightarrow Q_c'(z) = 2z$$

Das Einsetzen des ersten Punktes zeigt, daß zumindest z_1 abstoßend ist

$$|Q_c'(z_1)| = |2z_1| = 2\sqrt{1.408^2 + 0.275^2} = 2.869 > 1$$

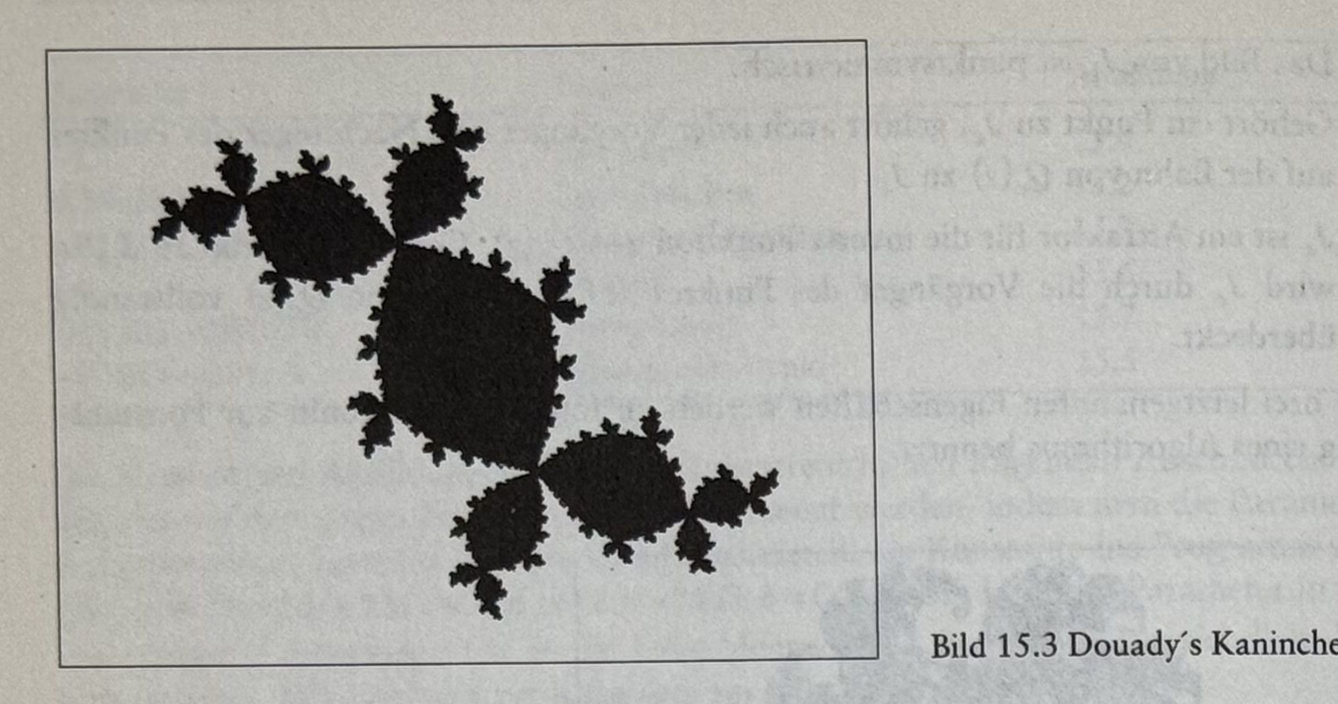


Bild 15.3 Douady´s Kaninchen

Solche abstoßenden Punkte liegen sogar dicht in J_c. Wegen dieser Eigenschaften rechnet man die Iteration $w \rightarrow z^2 + c$ rückwärts.

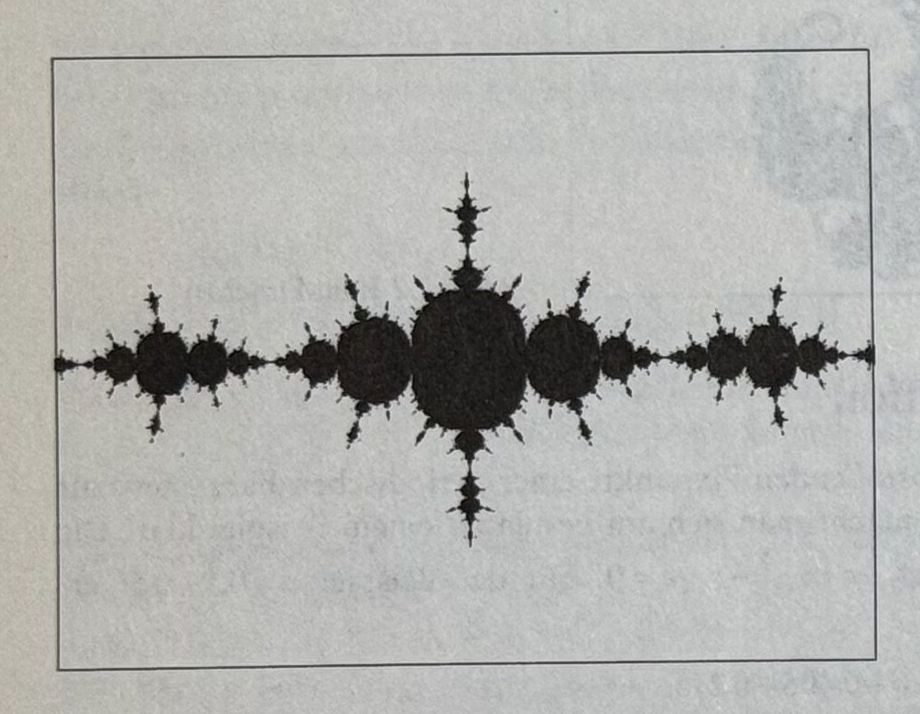


Bild 15.4 San Marco-ähnliche Julia-Menge

Dabei ergibt sich allerdings das Problem der nicht eindeutigen Umkehrbarkeit. Löst man die Gleichung $w = z^2 + c$ nach z auf, so ergibt sich $z = \pm\sqrt{w-c}$

Man behilft sich so, daß man mit Hilfe eines Zufallszahlen-Generators auslost, welche der komplexen Wurzeln gewählt wird. Die komplexe Wurzel kann mittels Polarkoordinaten berechnet werden:

$$\sqrt{z} = \left(re^{i\varphi}\right)^{\frac{1}{2}} = \pm\sqrt{r}e^{i\frac{\varphi}{2}}$$

Der so erhaltene Algorithmus heißt inverse Iterations-Methode (IIM). Er wird durch das folgende Turbo Pascal-Programm `juliim.pas` beschrieben.

```
program juliim;
{ Inverse Iterations-Methode für Julia-Mengen }
uses crt,graph;

const a = -1; b = 0; { Parameter der Julia-Menge }
var r,x,y,wx,wy,theta : real;
    i : integer;
    p,q : integer;
    Graphdriver,Graphmode:integer;

begin
GraphDriver := Detect;
Initgraph(GraphDriver,GraphMode,'c:\tp\bgi');
SetgraphMode(Graphmode);
rectangle(0,0,639,479);
moveto(8,8);outtext('Juliamenge J(-1,0)');

x :=0; y:=0;
for i:=1 to 32000 do
  begin
  wx := x-a;
  wy := y-b;
  if wx>0
       then theta := arctan(wy/wx)
  else if wx<0
       then theta := arctan(wy/wx)+pi
       else theta := pi/2;
  theta := theta/2;
  r := sqrt(wx*wx+wy*wy);
  if random<0.5 then r := sqrt(r) else r := -sqrt(r);
  x := r*cos(theta);
  y := r*sin(theta);
  p := 320+round(x*150); q := 240-round(y*150);
  if i>20 then
  begin
  putpixel(p,q,11); putpixel(640-p,480-q,11);
  if b=0 then
  begin putpixel(640-p,q,11); putpixel(p,480-q,11) end
  end;
 end;
repeat until keypressed;
closegraph;
textmode(lastmode)
end.
```

Die Real- und Imaginärteile a,b des Parameters sind im Programm als Konstante definiert. Beim Punktesetzen wurde hier von der Punktsymmetrie der Julia-Mengen Gebrauch gemacht. Verschwindet der Imaginärteil des Parameters, so kommt noch eine Achsensymmetrie dazu.

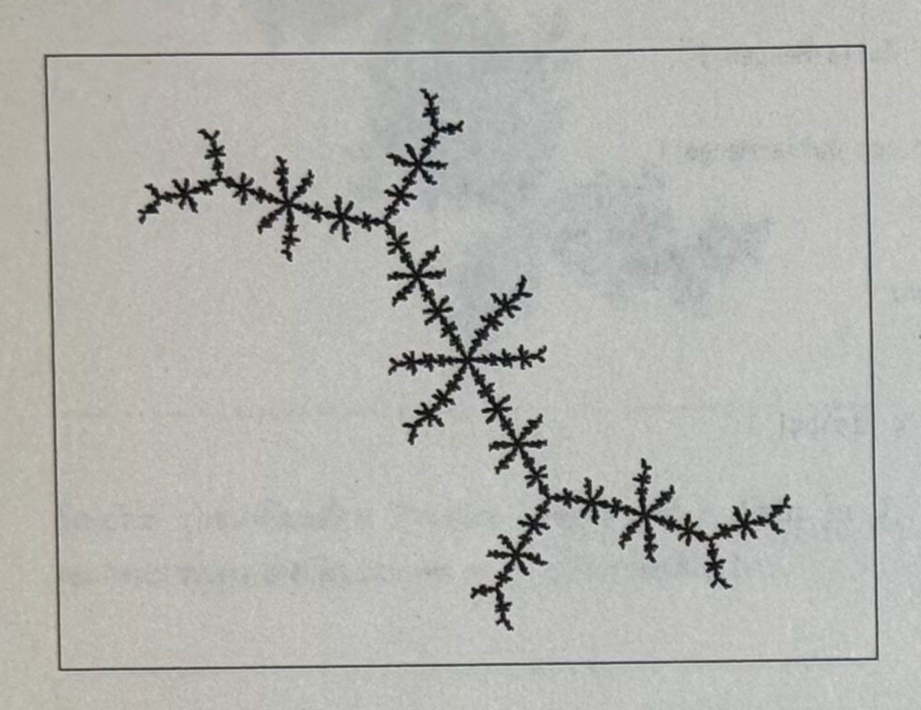


Bild 15.5 Julia-Menge an einem Misiurewicz-Punkt

Hier wird das komplexe Wurzelziehen algebraisch durchgeführt. Ist $w = u + vi$ die Wurzel von $z = x + yi$, so gilt $w^2 = z$. Trennung in Real- und Imaginärteil zeigt

$$u^2 - v^2 = x \wedge 2uv = y$$

Ebenfalls läßt sich die Gleichung $u^2 + v^2 = \sqrt{x^2 + y^2}$ herleiten. Auflösen liefert den gesuchten Real- und Imaginärteil der Wurzel

$$u = \pm\sqrt{x + \sqrt{\tfrac{1}{2}(x^2 + y^2)}}$$

$$v = \pm\sqrt{-x + \sqrt{\tfrac{1}{2}(x^2 + y^2)}}$$

15.3 Modifizierte inverse Iteration

Bei einigen Parameterwerten werden die Punkte der Julia-Menge nicht mit gleicher Wahrscheinlichkeit gezeichnet. Man führt daher ein Backtracking-Verfahren ein, das den Besuch der meisten Punkte sicherstellt. Zur Speicherung der Punkte wird ein zweidimensionales Feld benützt.

Das Verfahren ist im Quick Basic-Programm `julbak.bas` implementiert.

```
'julbak.bas
'Backtracking für Julia-Mengen

CONST a = -.2: b = .74: 'Parameter der Julia-Menge
CONST p = 64: 'Backtracking-Tiefe
CONST m1 = 200: m2 = 160: 'Auflösung
DIM a1, b1, d1, d2, r1, x, xd, x0, x1, y, yd, y0, y1 AS SINGLE
DIM g, m, n1, n2 AS INTEGER
DIM x(128), y(128), z(128), s(128)
DIM t(1 TO m1, 1 TO m2) AS INTEGER

SCREEN 12: CLS
WINDOW (-2, -1.5)-(2, 1.5)
LINE (-2, -1.5)-(2, 1.5), , B

d1 = 4 / m1: d2 = 3 / m2
a1 = 1 / 8 - a / 2: b1 = -b / 2
r1 = SQR(a1 * a1 + b1 * b1)
xd = 1 / 2 + SQR(r1 + a1): yd = SQR(r1 - a1)
IF b > 0 THEN yd = -yd
x = -xd: y = -yd: m = 0: g = 0
GOSUB graph
DO
  n1 = INT((2 - x) / d1): n2 = INT((1.5 - y) / d2)
  IF g < p AND t(n1, n2) < 20 THEN
      m = m + 1: g = g + 1: GOSUB trafo
      t(n1, n2) = t(n1, n2) + 1
      x = x1: y = y1
      x(m) = x2: y(m) = y2: s(m) = g
  ELSE
      x = x(m): y = y(m): g = s(m)
      m = m - 1
  END IF
  LOOP UNTIL m < 0
LOCATE 2, 2: PRINT "Julia-Menge J(-2,.74)"
e$ = INPUT$(1): SCREEN 0
END
trafo:
   x0 = (x - a) / 2: y0 = (y - b) / 2
   r = SQR(x0 * x0 + y0 * y0)
   x1 = -SQR(r + x0): y1 = -SQR(r - x0)
   IF y0 < 0 THEN y1 = -y1
   x2 = -x1: y2 = -y1
graph:
   PSET (x, y), 13: PSET (-x, -y), 13
   IF b = 0 THEN PSET (x, -y), 13: PSET (-x, y), 13
RETURN
```

15.4 Escape-Time-Algorithmus

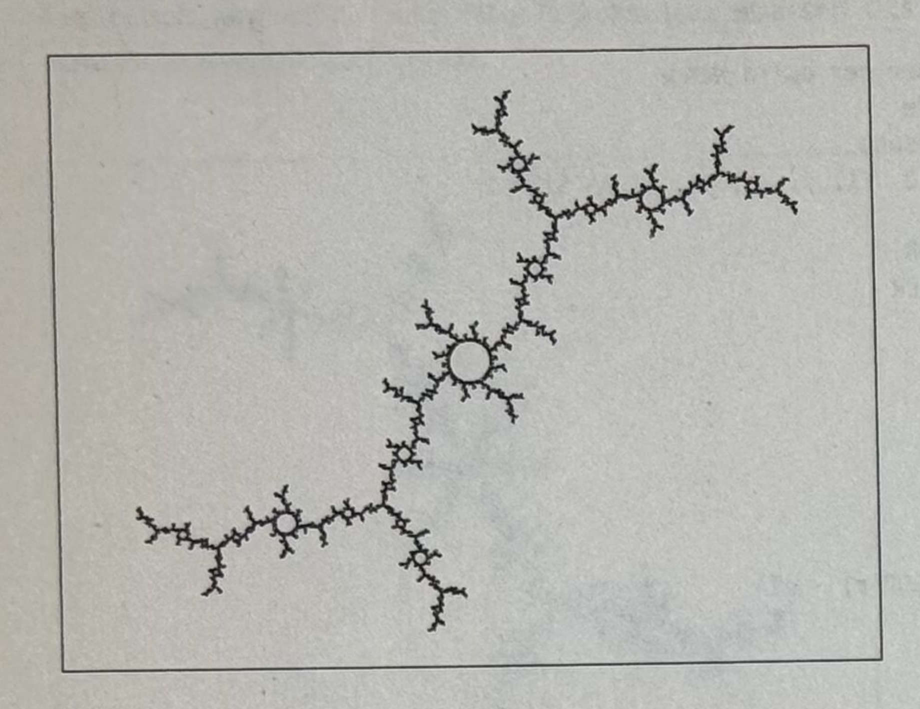


Bild 15.6 Dentrit

Ein Algorithmus, der eine Einfärbung einer Julia-Menge erlaubt, ist der sog. *Escape-Time*-Algorithmus. Für alle Punkte eines bestimmten Rechtecks wird die quadratische Iteration gestartet. Im Lauf der Rechnung wird entschieden, ob der Punkt zur Julia-Menge gehört. Dies ist der Fall, wenn die Bahn des Punkts nach Unendlich geht oder sich einem Fixpunkt nähert. Färbt man die Punkte entsprechend der Schrittzahl ein, so erhält man ein Farbbild der Julia-Menge. Um komplexe Arithmetik zu vermeiden, zerlegt man die komplexe Funktion $z \to z^2 + c$ in Real- und Imaginärteil. Mit $z = x + yi$ und $c = a + bi$ ergibt sich die Zerlegung

$$x \to x^2 - y^2 + a; \quad y \to 2xy + b$$

Eine Realisierung des Algorithmus zeigt das Quick Basic-Programm `julia.bas`.

```
'julia.bas
'Escape-time-Algorithmus für Julia-Mengen

CONST a = -.12: b = .74: 'Parameter der Julia-Menge
CONST xmax = 1.6: ymax = 1.2: 'Fenster
CONST n1 = 320: n2 = INT(n1 * ymax / xmax): 'Auflösung
CONST kmax = 200
DIM i, i1, i2, j, j1, j2, k AS INTEGER
DIM x, y, s1, s2, s3 AS DOUBLE
DIM col(15)

SCREEN 12: CLS
FOR i = 0 TO 15: READ col(i): NEXT i
IF b = 0 THEN n3 = 0 ELSE n3 = n2
```

```
FOR i = 0 TO n1
FOR j = -n3 TO n2
  i1 = 320 + i: i2 = 320 - i:
  j1 = 240 - j: j2 = 240 + j
  x = i * xmax / n1: y = j * ymax / n2
  FOR k = 1 TO kmax
    x1 = x * x - y * y + a: y1 = 2 * x * y + b
    s = x * x + y * y
    s1 = (x - x1) * (x - x1) + (y - y1) * (y - y1)
    IF s > 1000 THEN l = k MOD 16: GOTO graphik
    IF s1 < .0001 THEN l = k MOD 16: GOTO graphik
    x = x1: y = y1
  NEXT k
  l = 0
graphik:
  PSET (i1, j1), col(l): PSET (i2, j2), col(l)
  IF b = 0 THEN PSET (i1, j2), col(l): PSET (i2, j1), col(l)
NEXT j
NEXT i
e$ = INPUT$(1): SCREEN 0
END
DATA 0,1,9,4,12,2,1,9,4,12,2,1,9,4,12,2
```

15.5 Distanz-Verfahren

Ein sehr effizienter Algorithmus ist die sog. Distanz-Methode. Dabei wird das Einfärben der Punkte gemäß der Distanz $d(c,J)$ des Punktes von der Julia-Menge durchgeführt.

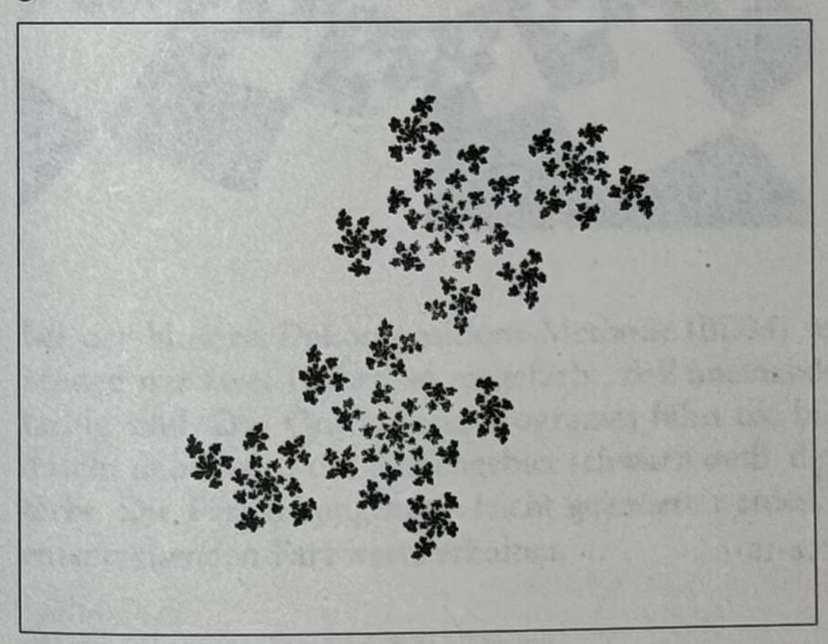


Bild 15.7 Fatou-Staub

Diese Distanz kann näherungsweise abgeschätzt werden durch

$$d(c,J) = \frac{|z_n| \ln |z_n|}{|z'_n|}$$

dabei ist z_n die Bahn des Punktes und z'_n seine Ableitung. Die Ableitungen werden im Programm ermittelt. Die Distanz-Methode ist im Turbo C-Programm `juldist.c` implementiert.

```
/* juldist.c */
/* Distanz-Mathode für Julia-Mengen */

#include <stdio.h>
#include <graphics.h>
#include <conio.h>
#include <stdlib.h>
#include <math.h>

const double a=-0.11031,b=0.67037;
const double xmax=1.4,ymax= 1.4;

void main(void)
{
double eps,dist,s1,s2,s3,x,x1,y,y1,u,u1,v,v1;
int i,j,k,n1,n2;
int gdriver,gmode;
gdriver = 9; gmode = VGAHI;
initgraph(&gdriver,&gmode," ");
rectangle(0,0,639,479);

n1 = 250; n2 = (int) (n1 * ymax / xmax);
eps = xmax/n1;
for (i=0; i<=n1; i++)
for (j=-n2; j<=n2; j++)
  {
  if (i==0 && j==0) break;
  x = i * eps; y = j * ymax/n2;
  u = 1; v = 0;
  for (k = 0; k<=200; k++)
    {
    x1 = x * x - y * y + a;
    y1 = 2 * x * y + b;
    u1 = 2 * (u * x - v * y);
    v1 = 2 * (u * y + v * x);
    s1 = x1 * x1 + y1 * y1 + 1.e-10;
    s2 = log(s1);
    s3 = sqrt(u1 * u1 + v1 * v1 + 1.e-10);
    if ((s1 > 50.) || (s3 > 50.))
      {
      dist = sqrt(s1) * s2 / s3;
      if (dist < eps)
```

```
            {
            putpixel(320+i,240-j,1+k/20);
            putpixel(320-i,240+j,1+k/20);
            }
         break;
         }
         x = x1; y = y1; u = u1; v = v1;
      }
}
moveto(8,8);outtext("Julia-Menge J(-0.11031,0.67037)");
do {} while(!(kbhit()));
closegraph();
return;
}
```

15.6 Binäre Dekompositions-Methode

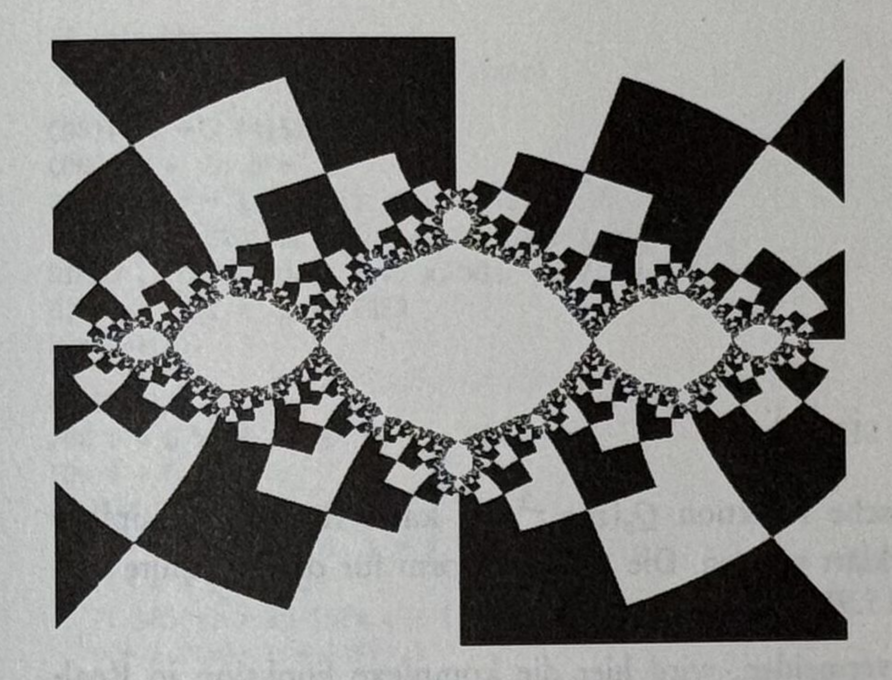


Bild 15.8 Binäre Dekomposition von San Marco

Bei der binären Dekompositions-Methode (BDM) wird der Außenraum einer Julia-Menge mit zwei Farben so eingefärbt, daß aneinanderstoßende Gebiete verschiedenfarbig sind. Das Quick Basic-Programm führt die binären Dekompositions-Methode durch; dabei wird das Außengebiet schwarz-weiß, die gefüllte Juliamenge weiß eingefärbt. Die Farbgebung kann leicht geändert werden. Dazu muß die Variable `col` die entsprechenden Farbwerte erhalten.

```
'julbin.bas
'Binär-Dekomposition von Julia-Mengen
CONST a = -1!: b = 0!:  'Parameter der Julia-Menge
```

```
CONST xmax = 1.8: ymax = 1.6: 'Bildschirmfenster
CONST n1 = 320: n2 = 240: 'Auflösung

SCREEN 12: CLS

FOR i = -n1 TO n1
FOR j = -n2 TO n2
  i1 = 320 + i: x = i * xmax / n1
  j1 = 240 - j: y = j * ymax / n2
  FOR k = 1 TO 100
     s = x * x + y * y
     IF (s > 100) THEN
        IF y > 0 THEN col = 0 ELSE col = 15
        GOTO graphik
     END IF
     x1 = x * x - y * y + a: y1 = 2 * x * y + b
     s1 = (x - x1) * (x - x1) + (y - y1) * (y - y1)
     IF s1 < .0001 THEN GOTO naechst
     x = x1: y = y1
  NEXT k
  col = 15
graphik:
  PSET (i1, j1), col
naechst:
NEXT j
NEXT i
LINE (0, 0)-(639, 479), , B
e$ = INPUT$(1): SCREEN 0
END
```

15.7 Julia-Menge z = sin(z)

Ebenso wie für die quadratische Funktion $Q_c(z) = z^2 + c$, kann für jede komplexe Funktion eine Julia-Menge erklärt werden. Die Iterationsform für das gewählte Beispiel heißt hier $z = \sin z$ (Bild 15.9).

Um komplexe Rechnung zu vermeiden, wird hier die komplexe Funktion in Real- und Imaginärteil zerlegt. Es gilt mit $z = x + yi$

$$\Re(\sin z) = \sin x \cosh y; \quad \Im(\sin z) = \cos x \sinh y$$

Die hyperbolischen Funktionen werden dabei über die Exponentialfunktion ermittelt

$$\sinh = \frac{1}{2}(e^x - e^{-x}); \quad \cosh x = \frac{1}{2}(e^x + e^{-x})$$

Der Escape-Time-Algorithmus wird für jedes Bildschirm-Pixel gestartet. Die Iteration bricht ab, wenn der Betrag des Realteils 20 überschreitet oder Konvergenz eintritt. Letzteres wird angenommen, wenn der Abstand eines Punktes von seinem Urbild kleiner als 10^{-6} ist. Eine Realisierung zeigt das Quick Basic-Programm `julsin.bas`.



Bild 15.9 Julia-Menge der Funktion z=sin(z)

```
'julsin.bas
'Julia-Menge der Gleichung z=sin(z)

CONST PI = 3.141529653#
CONST a = .6: b = .6
CONST xmax = 1.5 * PI: ymax = 3.5
CONST n1 = 320: n2 = INT(n1 * ymax / xmax)
DIM ch, cs, sh, d, ss, v, x, x1, x2, y, y1, y2 AS DOUBLE
DIM c, i, j, k AS INTEGER
DIM col(8)

SCREEN 12: CLS
FOR i = 0 TO 8: READ col(i): NEXT i
FOR i = 0 TO n1
   FOR j = -n2 TO n2
   x = i * xmax / n1: y = j * ymax / n2
   FOR k = 0 TO 50
   IF ABS(y) > 20 THEN c = 1 + k MOD 8: GOTO graphik
   u = EXP(y): v = EXP(-y)
   ch = (u + v) / 2: sh = (u - v) / 2
   cs = COS(x): ss = SIN(x)
   x1 = ss * ch: y1 = cs * sh
   x2 = a * x1 - b * y1: y2 = a * y1 + b * x1
   d = (x2 - x) * (x2 - x) + (y2 - y) * (y2 - y)
   IF d < .000001 THEN c = 1 + k MOD 8: GOTO graphik
   x = x2: y = y2
   NEXT k
   c = 0
graphik:
   PSET (320 + i, 240 - j), col(c): PSET (320 - i, 240 + j), col(c)
   NEXT j
NEXT i
```

```
e$ = INPUT$(1): SCREEN 0
END
DATA 0,1,9,2,10,4,12,14,6
```

15.8 Juliamenge z = exp(z)+C

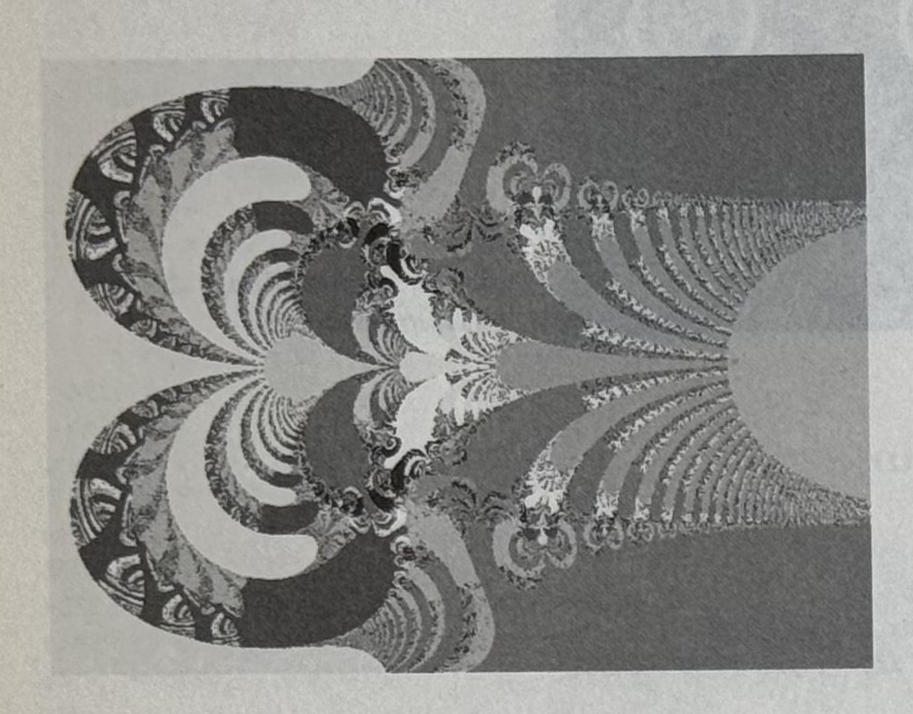


Bild 15.10 Julia-Menge der Funktion z=exp(z)

Als Beispiel einer komplexen Exponentialfunktion wird die Gleichung

$$z = e^z + C$$

gewählt (vgl. Bild 15.10). Da in der Programmiersprache C++ bereits eine komplexe Arithmetik implementiert ist, soll im folgenden ein entsprechendes Programm gegeben werden. Die Konstante C der Gleichung wird mittels Real- und Imaginärteil als konstant deklariert. Als Header muß hier zusätzlich die Datei `complex.h` eingeschlossen werden. Die vordefinierte Konstante `M_PI` ist in der Header-Datei `math.h` definiert und stellt die Zahl π in doppelter Genauigkeit dar.

Den Escape-Time-Algorithmus – in Form einer separaten Prozedur – zeigt das Turbo C++ Programm `julexp.cpp`.

```
// julexp.cpp
// Julia-Menge der Gleichung z=exp(z)+c

#include <graphics.h>
#include <stdlib.h>
#include <stdio.h>
#include <conio.h>
#include <math.h>
```

```
#include <complex.h> // Komplexe Arithmetik

void iteration(void);
int escape_time(double re,double im);
const double a=0.,b=0.; // Komplexe Konstante c=a+bi
const double xmin=-3.5,ymin=-3.,xmax=5.5,ymax=3.;
complex c;

void main(void)
{
int gdriver,gmode;
gdriver = 9; gmode = VGAHI; // VGA-Modus 640 x 480
initgraph(&gdriver,&gmode," ");

c = complex(a,b); // Konstante
iteration();
do {} while (!(kbhit()));
closegraph();
return;
}

void iteration(void)
{
int count;
double dx,dy,re,im;
dx = (xmax-xmin)/640.;
dy = (ymax-ymin)/480.;
re = xmin;
for (int i=1; i<=640; i++)
  {
  im = ymin;
  for (int j=1; j<=240; j++)
    {
    count = escape_time(re,im);
    if (count<=50)
      {
      putpixel(i,480-j+1,1+count % 15),
      putpixel(i,j,1+count % 15);
      }
     im += dy;
    }
  re += dx;
  }
}

int escape_time(double re,double im)
{
int count=0;
double x,y;
complex z;
z = complex(re,im);
while (count<=50 && real(z)<=50.)
    {
```

```
    count++;
    if ((fabs(imag(z)) > 1e4) || (fabs(real(z)) > 1e4))
      {
      x = imag(z);
      while (fabs(x) >= 2*M_PI)  x /= (2*M_PI);
      y = real(z);
      while (fabs(y) >= 2*M_PI)  y /= (2*M_PI);
      z = complex(x,y);
      }
    z = exp(z) + c;
    }
  return count;
  }
```

15.9 Newton-Verfahren

Das Newton-Verfahren zur Nullstellenbestimmung von reellen Funktionen wurde 1879 von A.Cayley auf die komplexe Zahlenebene erweitert. Für eine komplexe Funktion lautet die Iteration

$$z \to z - \frac{f(z)}{f'(z)}$$

Für das hier gewählte Beispiel $f(z) = z^3 - 1$ (Bild 15.11 und Farbtafel 9) ergibt sich die Iterationsform

$$z = z - \frac{z^3 - 1}{3z^2} = \frac{2z^3 + 1}{3z^2}$$

Die Lösungen der Gleichung $f(z) = z^3 - 1$ sind die dritten Einheitswurzeln

$$z_k = e^{\frac{2}{3}\pi i k};\ k = 0,1,2 \text{ oder } z_0 = 1;\quad z_{1,2} = -\tfrac{1}{2} \pm \tfrac{1}{2}\sqrt{3}i$$

Jede Einheitswurzel ist ein anziehender Fixpunkt mit einem Anziehungsbereich $A(z_k);\ k = 0,1,2$. Mit Hilfe eines Computerprogramms läßt sich ein solcher Anziehungsbereich sichtbar machen, indem man alle Startpunkte der Iteration, die gegen denselben Attraktor konvergieren, mit gleicher Farbe einfärbt. Hier zeigt sich überraschenderweise, daß alle drei Attraktionsgebiete gleich sind. Die Juliamenge ist damit der Rand aller drei Anziehungsbereiche $J_f = \partial A(z_k)$ mit $\partial A(z_1) = \partial A(z_2) = \partial A(z_3)$. Es ergibt sich folgender merkwürdige Sachverhalt: Sind zwei Attraktionsgebiete benachbart, so ist in einer Umgebung auch der dritte Anziehungsbereich zugegen. Peitgen und Richter haben diesen Sachverhalt in [22] wie folgt veranschaulicht:

Drei Mächte wollen einen Planeten unter sich aufteilen. Die Grenzziehung soll so erfolgen, daß überall dort, wo zwei Mächte eine gemeinsame Grenze haben, auch die dritte Macht zu-

gegen sein soll, um möglicherweise bei einem Konflikt eingreifen zu können. Die mit dem Newton-Verfahren gewonnene Einfärbung löst die Aufteilung der Territorien.

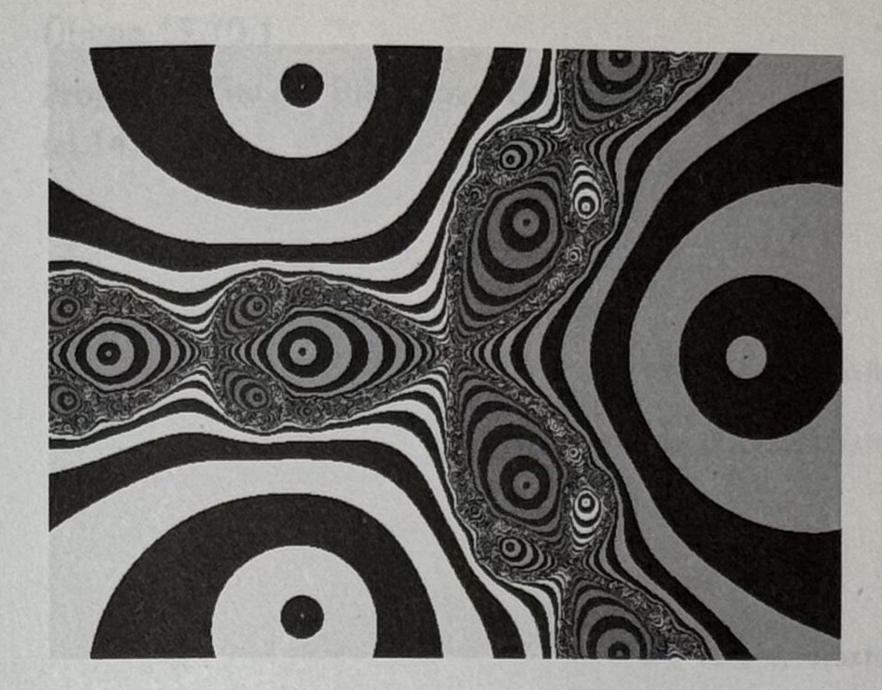


Bild 15.11 Juliamenge der Funktion z=z^3-1

Das Turbo Pascal-Programm `newton.pas` führt das komplexe Newton-Verfahren für die angegebene Funktion durch.

```
program newton;
{ Komplexes Newton-Verfahren für die Gleichung z^3-1=0 }
uses crt,graph;

label 111;
const ax = 1; ay = 0; { a,b,c 3.Einheitswurzeln }
      bx = -0.5; by = 0.8660254;
      cx = -0.5; cy = -0.8660254;
      eps = 0.005;
      xmin = -1.33333; xmax = 1.33333; { Bildschirmfenster}
      ymin = -1.0; ymax = 1.0;
var dx,dy,x,y,x1,y1,x2,y2,nenn:real;
    a,b,i,j,k:integer ;
    Graphdriver,Graphmode:integer;

function abstand(x,y,a,b:real):real;
begin
  abstand := sqrt(sqr(x-a)+sqr(y-b));
end;

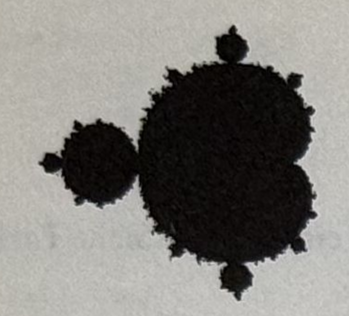
begin
GraphDriver := Detect;
Initgraph(GraphDriver,GraphMode,' ');
SetgraphMode(Graphmode);
```

```
a := getmaxx; b:= getmaxy;
dx := (xmax-xmin)/a;
dy := (ymax-ymin)/b;
for i := 0 to a-1 do
for j := 0 to b div 2 do
begin
k := 0;
x := xmin+i*dx;
y := ymin+j*dy;
repeat
   if abstand(x,y,ax,ay)<eps then
   begin
   if odd(k) then
      begin putpixel(i,j,1); putpixel(i,b-j,1) end
   else
      begin putpixel(i,j,9); putpixel(i,b-j,9) end;
   goto 111;
   end;
   if abstand(x,y,bx,by)<eps then
   begin
   if not odd(k) then
      begin putpixel(i,j,2); putpixel(i,b-j,2) end
   else
      begin putpixel(i,j,10); putpixel(i,b-j,10)end;
   goto 111;
   end;
   if abstand(x,y,cx,cy)<eps then
   begin
   if odd(k) then
      begin putpixel(i,j,6); putpixel(i,b-j,6) end
   else
      begin putpixel(i,j,14); putpixel(i,b-j,14) end;
   goto 111;
   end;
   if abs(x-y)<0.0001 then goto 111;
   x2 := x*x; y2 := y*y;  nenn := (x2+y2)*(x2+y2);
   x1 := (2.0*x+(x2-y2)/nenn)/3.0;
   y1 := 2.0*(y-x*y/nenn)/3.0;
   x := x1; y := y1;
   inc(k)
until k=25;
111:end;
repeat until keypressed;
closegraph;
textmode(lastmode);
end.
```

15.10 Ergänzung

Übung 15.10.1

Projizieren Sie die Juliamenge der Funktion $f(z) = z^3 - 1$ auf eine Kugel (Siehe Farbtafel 14).



16 Mandelbrot-Menge

Die Mandelbrot-Menge M ist die Menge aller komplexen Zahlen $c \in C$, für die der Grenzwert der iterierten Gleichungen der quadratischen Funktion

$Q_c(z) = z^2 + c$ mit dem Startwert $z = 0$ beschränkt bleibt.

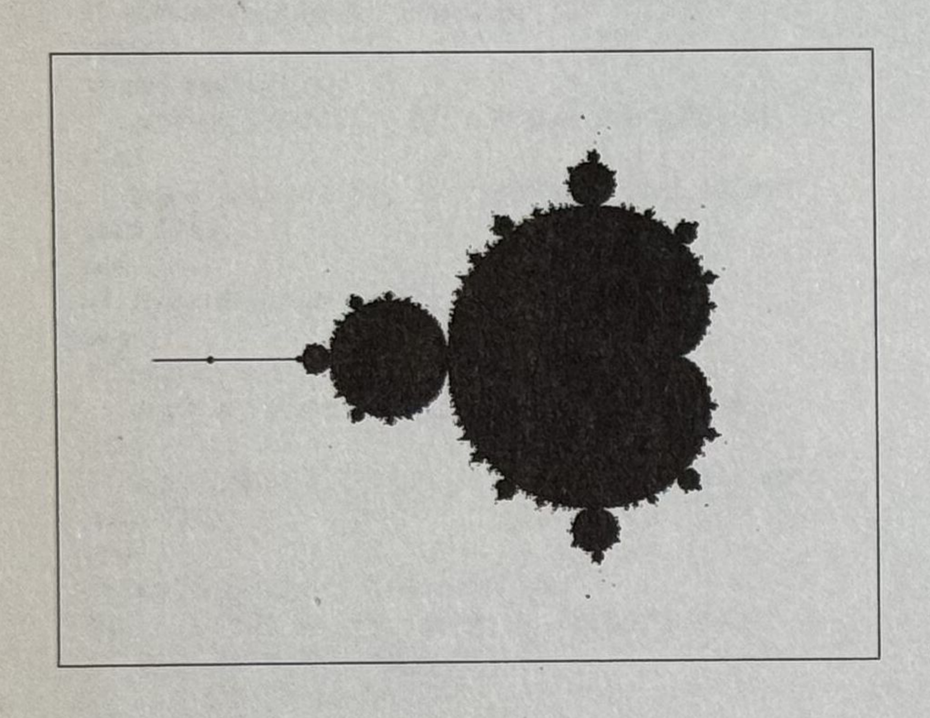


Bild 16.1 Mandelbrot-Menge, meist Apfelmännchen genannt

$$M = \left\{ c \in C;\ \lim_{n \to \infty} \left| Q_c^n(0) \right| < \infty \right\}$$

Die quadratische Funktion $Q_c(z) = z^2 + c$ ist holomorph; d.h. komplex differenzierbar. Dies sieht man am einfachsten an der Gültigkeit der Cauchy-Riemannschen Gleichungen. Zerlegt man die Funktion in den Real- und Imaginärteil, so folgt mit $c = a + bi$

$$\Re(Q_c) = u(x, y) = x^2 - y^2 + a;\ \Im(Q_c) = v(x, y) = 2xy + b$$

Die Jacobi-Matrix ist

$$\mathbf{J} = \begin{pmatrix} \frac{\partial u}{\partial x} & \frac{\partial u}{\partial y} \\ \frac{\partial v}{\partial x} & \frac{\partial v}{\partial y} \end{pmatrix} = \begin{pmatrix} 2x & -2y \\ 2y & 2x \end{pmatrix}$$

Es gilt somit die Gleichheit der partiellen Ableitungen

$$\frac{\partial u}{\partial x} = \frac{\partial v}{\partial y} \wedge \frac{\partial u}{\partial y} = -\frac{\partial v}{\partial x}$$

Dies zeigt, daß die Cauchy-Riemannschen Gleichungen erfüllt sind. Die Funktion $Q_c(z) = z^2 + c$ stellt somit eine winkeltreue Abbildung (für $c \neq 0$) dar. Sie bildet ein Gitter von parallelen waagrechten und senkrechten Geraden auf eine Schar von orthogonalen Parabeln ab.

16.1 Dynamik der Mandelbrot-Menge

Die Ableitung $Q_c'(z) = 2z$ verschwindet genau für $z = 0$. Der Ursprung ist somit ein superstabiler Fixpunkt der Abbildung. Für einen neutralen Fixpunkt erhält man die Gleichung

$$\left|Q_c'(z)\right| = |2z| = 1 \Rightarrow |z| = \tfrac{1}{2}$$

d.h. ein neutraler Fixpunkt liegt auf dem Kreis um den Ursprung mit Radius $\frac{1}{2}$. Schreibt man diesen Kreis in komplexer Form $z = \frac{1}{2}e^{i\varphi}$, so liefert das Einsetzen in die Fixpunkt-Gleichung $c = z - z^2$ die Bedingung

$$c = \frac{1}{2}e^{i\varphi} - \frac{1}{4}e^{2i\varphi}$$

Trennung der Real- und Imaginärteile liefert

$$a = \frac{1}{2}\cos\varphi - \frac{1}{4}\cos 2\varphi;\ b = \frac{1}{2}\sin\varphi - \frac{1}{4}\sin 2\varphi$$

An dieser Form erkennt man, daß diese Punktemenge eine Herzkurve oder Kardioide darstellt, die den »Bauch« des Apfelmännchens bildet. Sie schneidet die reelle Achse in den Punkten $c = \frac{1}{4} \wedge c = -\frac{3}{4}$.

Für die zweite Iterierte der Funktion folgt $F_c(z) = Q_c^2(z) = (z^2 + c)^2 + c$. Für Fixpunkte der Periode Zwei muß gelten $F_c(z) = z$ oder

$$z^4 + 2cz^2 - z + c^2 + c = (z^2 - z + c)(z^2 + z + c + 1)$$

Der erste Faktor dieser Zerlegung liefert die schon bekannte Fixpunkte der ersten Iterierten. Zerlegung des zweiten Faktors ergibt

$$z_{1,2} = \tfrac{1}{2}(-1 \pm \sqrt{-3 - 4c})$$

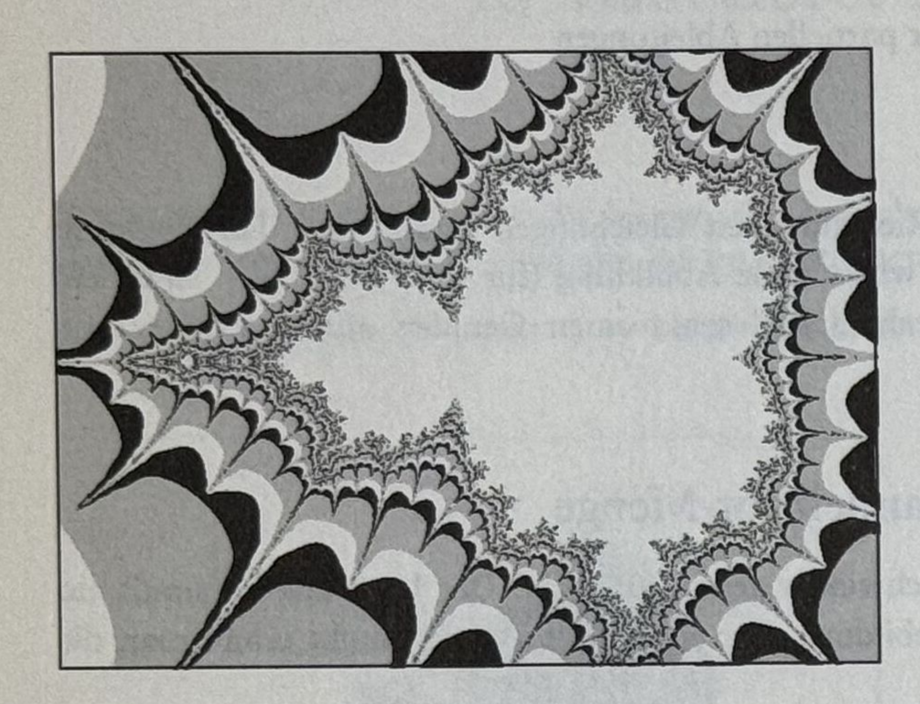


Bild 16.2 Ausschnitt aus der Antenne des Apfelmännchen bei z=-2

Mit der Ableitung $F'(z) = 4z^3 + 4cz$ erhält man $F'(z_1) = F'(z_2) = 4(1+c)$. Für die Bedingung eines neutralen Fixpunktes $|F_c'(z)| = 1$ ergibt sich

$$|4(1+c)| = 1 \Rightarrow |1+c| = \tfrac{1}{4}$$

Diese Punktmenge stellt einen Kreis mit dem Mittelpunkt -1 und dem Radius $\frac{1}{4}$ dar, der auch in der Form $c = \frac{1}{4}e^{2\pi i\varphi} - 1$ geschrieben werden kann. Dieser Kreis stellt den »Kopf« des Apfelmännchens dar.

Das Verfahren läßt sich in der angegebenen Weise fortsetzen. Für die dritte Iterierten erhält man Kreise, die die »Arme« und die »Antenne« des Apfelmänchens darstellen.

Es läßt sich zeigen, daß jede natürliche Zahl als Zyklus der Mandelbrot-Menge auftritt. Die Zyklen des Apfelmännchen lassen sich auch am Feigenbaum-Diagramm der reellen Funktion $Q_c(x) = x^2 + c$ ablesen (Bild 16.3).

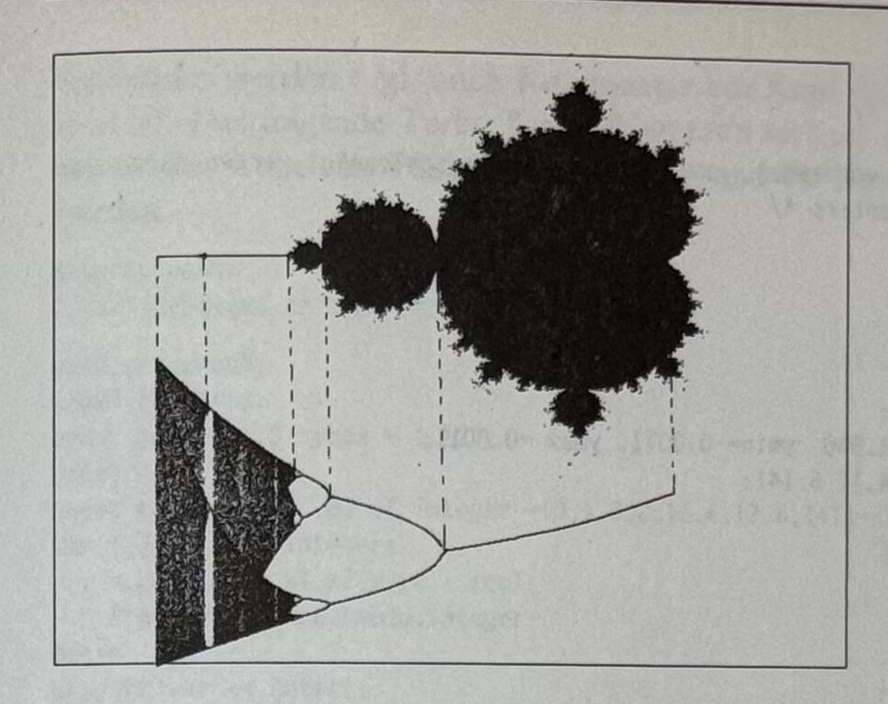


Bild 16.3 Periodizität des Apfelmännchens und Bifurkationen von x^2+c

16.2 Eigenschaften der Mandelbrot-Menge

- Die Mandelbrot-Menge ist symmetrisch bezüglich der reellen Achse. Ist nämlich die Bahn eines komplexen Punktes beschränkt, so gilt dies auch für die Bahn der konjugiert-komplexen Zahl.
- Die Schnittmenge mit der reellen Achse ist das Intervall $\left[-2;\frac{1}{4}\right]$. Ein Beweis dazu findet sich bei Zeitler[40].
- Die Mandelbrot-Menge ist beschränkt, sie liegt im Inneren des Kreises $|z|=2$.
- Die Mandelbrot-Menge ist selbstähnlich; jedoch ist diese Ähnlichkeit nicht exakt.
- Die Mandelbrot-Menge ist zusammenhängend. Dieses Ergebnis, daß scheinbar allen Apfelmännchen-Graphiken widerspricht, konnte erst 1982 von Douady/Hubbard bewiesen werden.
- Der Rand der Mandelbrot-Menge ist fraktal. Die Bestimmung der fraktalen Dimension ist noch Gegenstand mathematischer Forschung. Für die Hausdorff-Dimension D von J_c konnte D. Ruelle zeigen, daß für kleine c gilt:

$$D = 1 + \frac{|c|^2}{4 \lg 2} + O(c^3)$$

16.3 Escape-Time Algorithmus

Für jeden Punkt des Bildschirms wird die Iteration $Q_c(z) = z^2 + c$ gestartet, wobei aber im Gegensatz zur Julia-Iteration die Konstante der letzte Iterationswert ist. Das Verfahren kann dem Turbo C-Programm `apfel.c` entnommen werden.

```
/* apfel.c */
/* Apfelmännchen bzw. Ausschnitt */

/* Falls Symmetrie ymax == -ymin vorhanden, kann das Programm beschleunigt werden durch
Wegnahme des untenstehenden Kommentars */

#include <stdio.h>
#include <graphics.h>
#include <conio.h>
#include <stdlib.h>
const int kmax = 100,grenze = 10;
const double xmin=-1.943, xmax=-1.940, ymin=-0.0011, ymax =0.0011;
static int col[9] = {0,1,9,2,10,4,12,6,14};

void main(void)
{
register int i,j,k;
int a,b;
double h1,h2,x,y,x0,y0,dx,dy,p,q;
int gdriver,gmode;
gdriver = 9; gmode = VGAHI; /* VGA-Modus 640 x 480 */
initgraph(&gdriver,&gmode," ");
a = getmaxx(); b= getmaxy();

dx = (xmax-xmin)/a;
dy = (ymax-ymin)/b;
for (i=1; i<=a; i++)
for (j=1; j<=b; j++)
 /* Falls symmetrisch: for (j=1; j<=b/2; j++) */
  {
  p = xmin+i*dx; q = ymin+j*dy;
  k = 0; x = y = 0;
  do {
     h1 = x*x;  h2= y*y;
     x0 = h1 - h2 + p;
     y0 = 2*x*y + q;
     x = x0; y = y0; k++;
     }
  while(k<kmax && h1+h2<grenze);
  if (k<kmax)
     {
     putpixel(i,j,col[1+k % 8]);
/* Falls symmetrisch ergänzen: putpixel(i,b-j,col[1+k%8]); */
     }
  }
do {} while(!(kbhit()));
closegraph();
return;
}
```

Soll ein zur Realteil-Achse symmetrischer Ausschnitt des Apfelmännchens geplottet werden, so kann die Symmetriebeziehung zur Beschleunigung des Programmlaufs

verwendet werden (vgl. auch Kommentar am Kopf des vorherstehenden Programms `apfel.c`). Das folgende Turbo Pascal-Programm `mand.pas` zeichnet das vollständige Apfelmännchen, wobei die Punkte des »Kopfes« und die Kardiode des »Bauchs« ausgespart werden.

```
program mand;
{ Vollständiges Apfelmännchen }

uses crt,graph;
label naechst;
const xmax = 1.5; ymax = 1.125;
const n1 = 320;
const col:array[0..8] of integer =(0,1,9,2,10,4,12,6,14);
var i,j,k,l,n2 :integer;
    a,b,s,u,v,x,x1,x2,y,y2 : real;
    Graphdriver,Graphmode:integer;
begin
GraphDriver := Detect;
Initgraph(GraphDriver,GraphMode,' ');
SetgraphMode(Graphmode);
n2 := round(n1*ymax/xmax);
for i := -n1 to n1 do
begin
   a := -0.65 + i*xmax/n1;
   for j := 0 to n2 do
   begin
   b := j * ymax / n2;
   u := 4*(a * a + b * b); v := u-2*a+0.25;
   if (u+8*a+3.75 < 0) or (v-sqrt(v)+2*a-0.5 < 0) then goto naechst;
   x := a; y := b; k := 0;
   repeat
      x1 := x; k := k+1;
      x2 := x * x; y2 := y * y;
      x := x2 - y2 + a;
      y := 2 * x1 * y + b;
      s := x2 + y2;
   until (s>4) or (k=40);
   if k<40 then
   begin
   l := l+k mod 8;
   putpixel(320+i,240-j,col[l]);
   putpixel(320+i,240+j,col[l]);
   end;
   naechst:
   end
end;
repeat until keypressed;
closegraph;
textmode(lastmode)
end.
```

16.4 Distanz-Methode

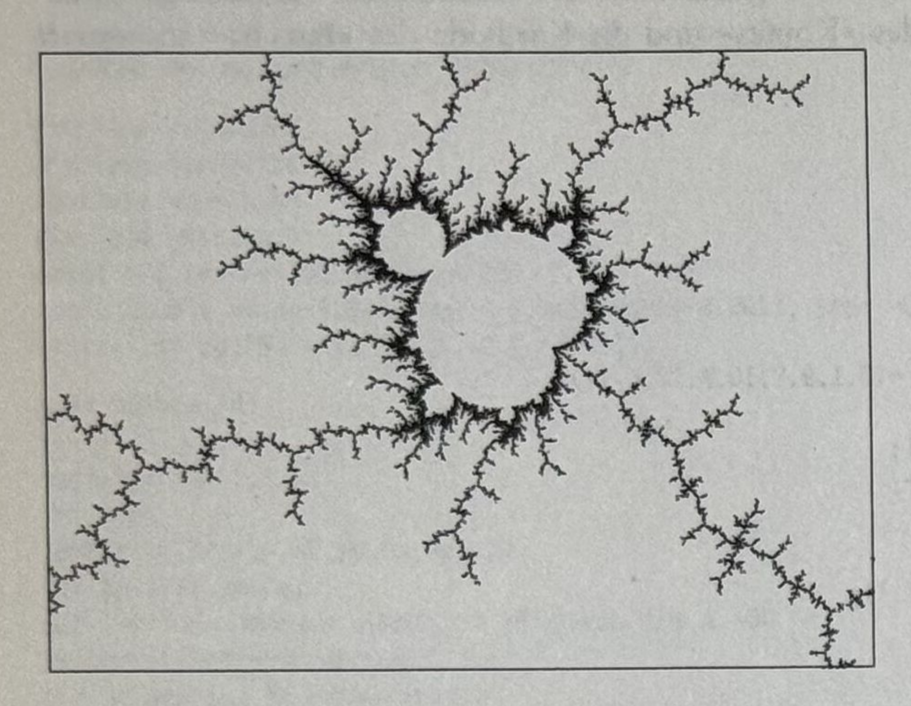


Bild 16.4
Ausschnittsvergrößerung des Apfelmännchens bei z=-0.16+1.03i

Das im Abschnitt 15.5 vorgestellte Distanzverfahren läßt sich analog auf die Mandelbrot-Menge anwenden. Dies zeigt das folgende Quick Basic-Programm mandist.bas.

```
'mandist.bas
'Mandelbrot-Menge mittels Distanzmethode

CONST ax = -.16: ay = 1.03: 'Mittelpunkt des Ausschnitts
CONST dx = .025: dy = .75 * dx: ' Ausschnittbreite
CONST n1 = 320: n2 = INT(.75 * n1)
CONST kmax = 200
DIM a, b, eps, s1, s2, s3, u, u1, v, v1, x, x1, y, y1 AS DOUBLE
DIM i, j, k AS INTEGER

SCREEN 12: CLS

eps = .4 * dx / n1
FOR i = -n1 TO n1
FOR j = -n2 TO n2
  a = ax + i * dx / n1: b = ay + j * dy / n2
  x = a: y = b: u = 1: v = 0
  FOR k = 0 TO kmax
    x1 = x * x - y * y + a: y1 = 2 * x * y + b
    u1 = 2 * (u * x - v * y) + 1: v1 = 2 * (u * y + v * x)
    s1 = x1 * x1 + y1 * y1 + 1E-08
    IF s1 > 128 THEN
      s2 = LOG(s1): s3 = SQR(u1 * u1 + v1 * v1)
      dist = SQR(s1) * s2 / s3
    IF dist < eps THEN PSET (320 + i, 240 - j), 1 + INT(100000! * dist)
```

```
      GOTO naechst
    END IF
    x = x1: y = y1: u = u1: v = v1
  NEXT k
naechst:
NEXT j
NEXT i
e$ = INPUT$(1): SCREEN 0
END
```

16.5 Mandelbrot-Menge einer Sinusfunktion

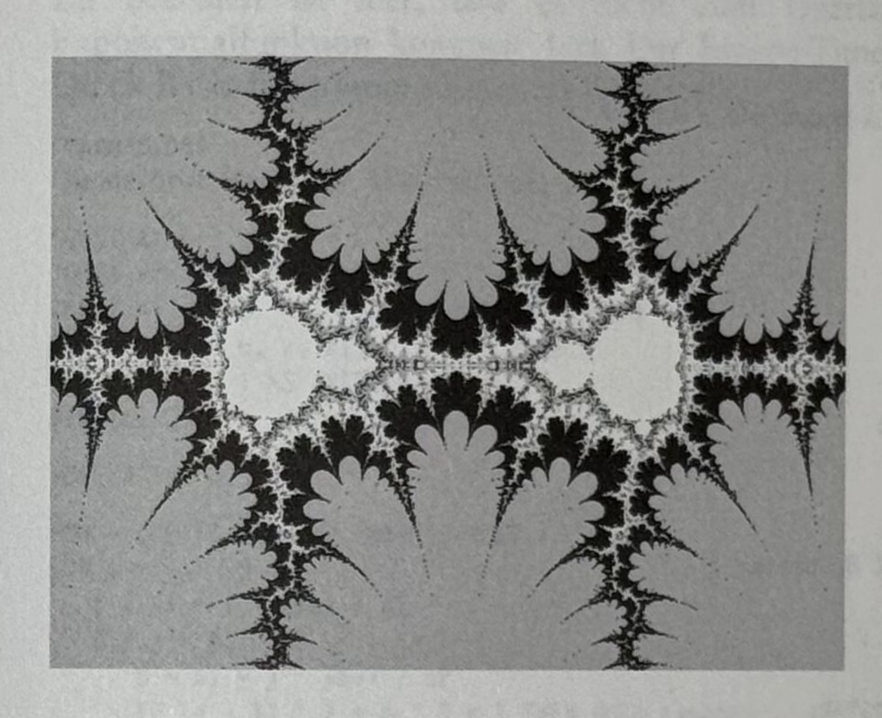


Bild 16.5 Mandelbrot-Menge von $z \rightarrow c+\pi \sin z$

Die bei der Mandelbrot-Menge verwendete Iteration kann auf einfache Weise auf andere Funktionen übertragen werden. Als Beispiel sei hier die komplexe Funktion

$$z \rightarrow c+\pi \sin z$$

gewählt (vgl. Bild 16.5 und Farbtafel 12). Um komplexes Rechnen zu vermeiden, wird die Sinus-Funktion in Real- und Imaginärteil zerlegt. Es gilt mit $z=x+yi$

$$\Re(\sin z)=\sin x \cosh y; \Im(\sin z)=\cos x \sinh y$$

Die benötigten hyperbolischen Funktionen werden hier über die Exponentialfunktion berechnet. Die Symmetrien zur Real- und Imaginärteil-Achse werden im Programm benützt.

```
/* mandsin.c */
/* Mandelbrot-Menge der Funktion c+pi*sin(z) */
```

```
#include <stdio.h>
#include <graphics.h>
#include <math.h>
#include <conio.h>

const double PI=3.141592653;
const int n1=320;
int col[9] = {0,1,9,2,10,4,12,14,6};

void main(void)
{
double a,b,u,v,x,y,dist,ch,cs,sh,ss,x1,y1;
double xmax=PI,ymax=0.75*PI;
int i,i1,i2,j,j1,j2,k,l;
int n2,gdriver,gmode;

n2 = (int)floor(n1*ymax/xmax+0.5);
gdriver = 9; gmode = VGAHI; /* VGA-Modus 640 x 480 */
initgraph(&gdriver,&gmode," ");

for (i=0; i<=n1; i++)
{
   i1 = 320+i; i2 = 320-i;
   for (j=0; j<=n2; j++)
   {
   j1 = 240+j; j2 = 240-j;
   a =  xmax*i/n1; b = ymax*j/n2;
   x = PI/2.;  y = 0;
   for (k=1; k<=64; k++)
     {
     if (fabs(y)>15.) { l = 1+k % 8; break; }
     u = exp(y); v = 1./u;
     ch = (u+v)/2.; sh = (u-v)/2.;
     ss = sin(x); cs = cos(x);
     x1 = a+PI*ss*ch; y1 = b+PI*cs*sh;
     dist = fabs(x-x1)+fabs(y-y1);
     if (dist<.0005) {l=0; break; }
     x = x1; y = y1; l=0;
     }
  putpixel(i1,j1,col[l]); putpixel(i1,j2,col[l]);
  putpixel(i2,j1,col[l]); putpixel(i2,j2,col[l]);
  }
}
do {} while(!(kbhit()));

closegraph();
return;
}
```

16.6 Mandelbrot-Menge einer Exponentialfunktion

Als Beispiel einer Exponentialfunktion soll hier die Funktion

$$f(z) = c + z - e^z$$

behandelt werden (Siehe Bild 16.6 und Farbtafel 13). Der Real- und Imaginärteil der Funktion ist

$$\Re(f(z)) = c_x + x - e^x \cos y; \Im(f(z)) = c_y + y - e^x \sin y$$

Zu beachten ist hier, daß es nicht zum Überlauf bei der Berechnung der Exponentialfunktion kommen darf. Der Escape-Time-Algorithmus wird durch das Quick Basic-Programm `mandexp.bas` dargestellt.

```
'manexp.bas
'Mandelbrot-Menge der Gleichung c+z-exp(z)

CONST ax = 3.8: ay = 0
CONST xmax = 3.8: ymax = 2.7
CONST n1 = 320: n2 = INT(n1 * ymax / xmax)
DIM a, b, d, u, x, x1, y, y1 AS DOUBLE
DIM i, j, k, l AS INTEGER
DIM col(8)

SCREEN 12: CLS

FOR i = 0 TO 8: READ col(i): NEXT i
FOR i = -n1 TO n1
   a = ax + i * xmax / n1
   FOR j = 0 TO n2
      b = ay + j * ymax / n2
      IF (a - 1) ^ 2 + b ^ 2 < 1 THEN GOTO naechst
      x = 0: y = 0
      FOR k = 1 TO 30
         IF x > 10 THEN l = 1 + k MOD 8: GOTO graphik
         IF x < -10 THEN u = 0 ELSE u = EXP(x)
         x1 = a + x - u * COS(y): y1 = b + y - u * SIN(y)
         d = (x - x1) ^ 2 + (y - y1) ^ 2
         IF d < .0001 THEN GOTO naechst
         x = x1: y = y1
      NEXT k: l = 0
graphik:
   PSET (320 + i, 240 - j), col(l): PSET (320 + i, 240 + j), col(l)
naechst:
   NEXT j
NEXT i
e$ = INPUT$(1): SCREEN 0
DATA 0, 1, 9, 2, 10, 6, 14, 4, 12
END
```

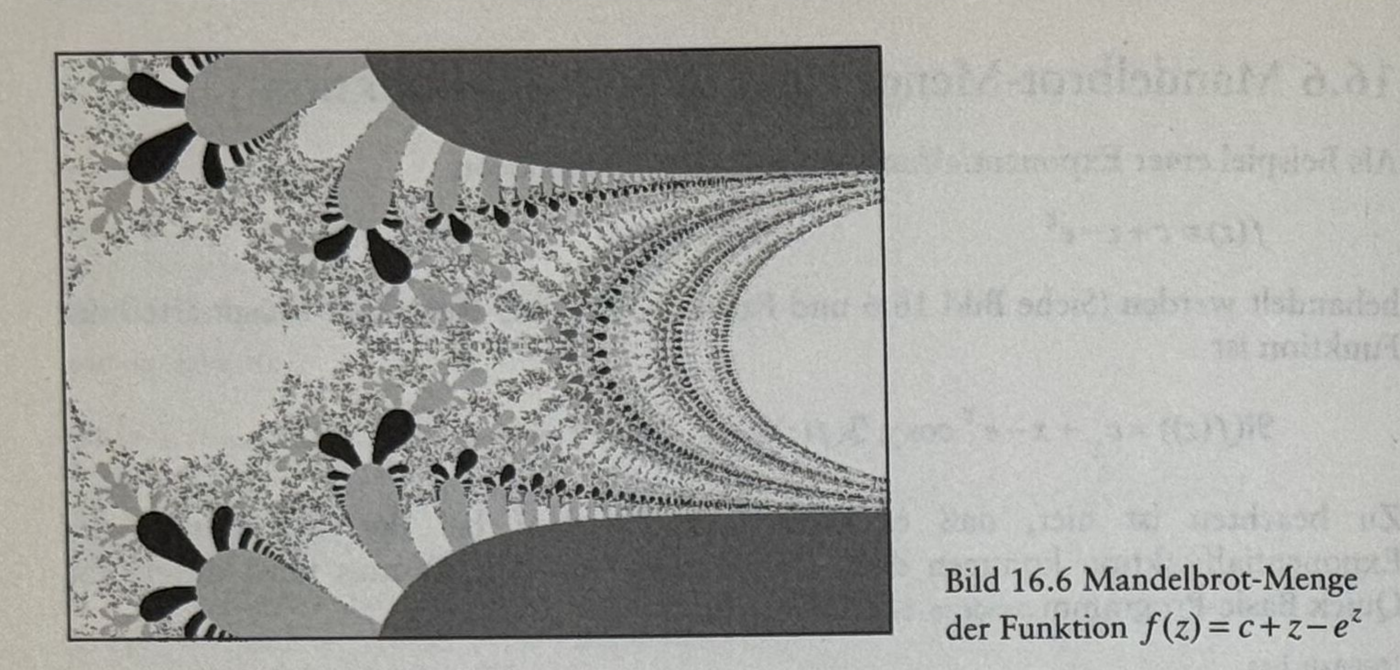
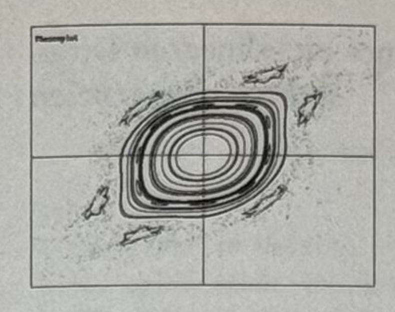


Bild 16.6 Mandelbrot-Menge der Funktion $f(z) = c + z - e^z$



17 Anwendungen I

In diesem Kapitel sollen einige Anwendungen der Chaostheorie in den verschiedenen Wissenschaften aufgezeigt werden. Hier kann natürlich keine Vollständigkeit angestrebt werden, allein die Aufzählung der bekannten physikalischen Anwendungen ist kaum möglich. Man vergleiche nur die Themenvielfalt in dem Sammelband *Fractals in Physics* [24]!

17.1 Chemie: Brüsselator

Eine der überraschendsten Entdeckungen in der Chemie der letzten Jahre war die Entdeckung der oszillierenden chemischen Reaktionen, bei denen eine Reaktion – fern vom Gleichgewichtspunkt – zwischen zwei Zuständen oszilliert.

Der Russe B. P. Belusow hatte 1958 bei einer Mischung von Malon-, Schwefelsäure, Kaliumbromat und Ceriumsulfat eine oszillierende Reaktion entdeckt. 1963 verbesserte sein Landsmann A. M. Zhabotinski die Reaktion durch Verwendung von Eisensulfat (statt des Ceriumsulfats) und erhielt spektakuläre Farbumschläge zwischen Blau und Rot, die man ca. 10 Minuten lang mit bloßem Auge beobachten konnte.

Die Belusow-Zhabotinski-(BZ)-Reaktion wird beschrieben durch die Bruttoformel

$$2BrO_3^- + 3CH_2(COOH)_2 + 2H^+ \rightarrow 2BrCH(COOH)_2 + 3CO_2\uparrow + 4H_2O$$

Der genaue Reaktionsablauf ist sehr kompliziert und erfolgt in über 20 Stufen. Die BZ-Reaktion konnte erst 1972 vollständig geklärt werden.

Für Reaktionen der Art, wie sie bei der BZ-Reaktion vorliegen

$$A \rightarrow X$$
$$B + X \rightarrow D + Y$$
$$2X + Y \rightarrow 3X$$
$$X \rightarrow E$$

lieferte Prigogine auch ein mathematisches Modell in Form eines nichtlinearen Differentialgleichungs-Systems. Für jede der beteiligten sechs Reagenzien A, B, D, E, X, Y ergibt sich eine Differentialgleichung.

$$\dot{A} = -k_1 A$$
$$\dot{B} = -k_2 BX$$
$$\dot{D} = k_2 BX$$
$$\dot{E} = k_3 X$$
$$\dot{X} = k_1 A - k_2 B - k_3 X + k_4 X^2 Y$$
$$\dot{Y} = k_2 BX - k_4 X^2 Y$$

Dabei sind k_i $(i = 1,2,3,4)$ chemische Reaktionskonstanten. Dieses System wird nach der Stadt, in der sich das Universitätsinstitut von Prigogine befindet, Brüsselator genannt.

Das Turbo Pascal-Programm `brussel.pas` integriert numerisch den Brüsselator. Die verwendeten Reaktionskonstanten und Anfangskonzentrationen sind dem Aufsatz *Oscillating Systems of Chemical Reactions* von H. Hoekstra und C. Kok in Junge Wissenschaft 30, 8 entnommen.

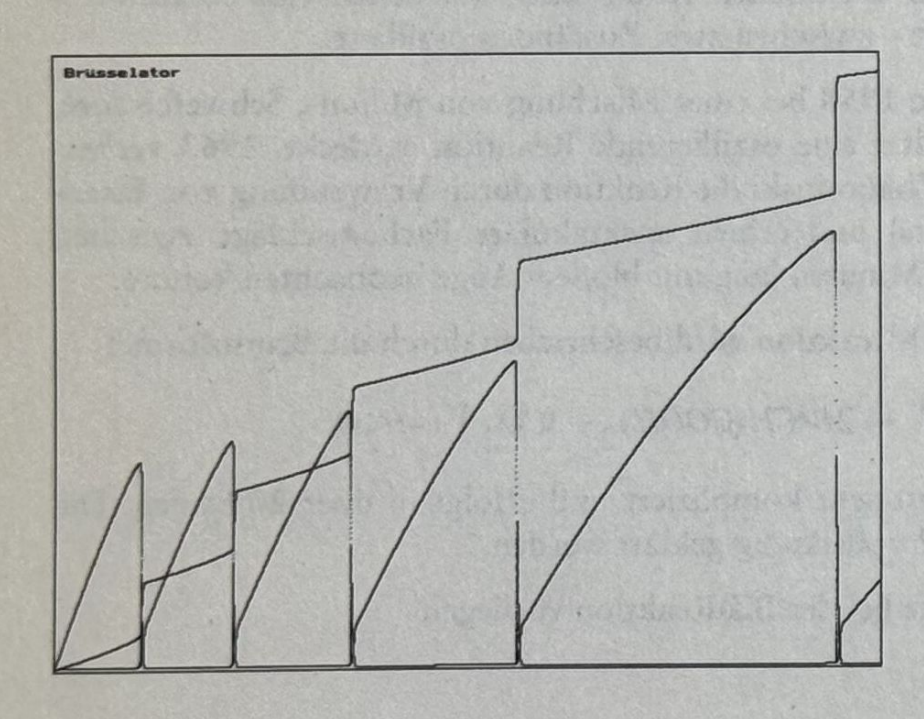


Bild 17.1 Oszillationen des Brüsselators

```
program brussel;
{$N+}
uses crt,graph;

const k1 = 0.0015; k2 = 0.00049; k3 = 0.877;k4 = 10;
```

```
    dt = 0.001;
var a,b,d,e,x,y,da,db,dd,de,dx,dy,t : double;
    i : longint;
    Graphdriver,Graphmode:integer;

begin
GraphDriver := Detect;
Initgraph(GraphDriver,GraphMode,' ');
SetgraphMode(Graphmode);
cleardevice;
rectangle(0,0,639,479);
moveto(10,10);outtext('Brüsselator');

a:=60 ; b:= 5000; d := 0; e:=0; x:=0; y:=0; { Anfangskonzentrationen }
t := 0;
while t<= 640 do
   begin
   da := -k1*a*dt;
   db := -k2*b*x*dt;
   dd := k2*b*x*dt;
   de := k3*x*dt;
   dx := (k1*a-k2*b*x-k3*x+k4*x*x*y)*dt;
   dy := (k2*b*x - k4*x*x*y)*dt;
   a := a+da;
   b := b+db;
   d := d+dd;
   e := e+de;
   x := x+dx;
   y := y+dy;
   putpixel(round(t),477-round(22*x),14);
   putpixel(round(t),477-round(45*y),12);
   putpixel(round(t),477-round(13*e),13);
   t := t+dt;
   end;
repeat until keypressed;
closegraph;
TextMode(lastmode)
end.
```

Die Oszillationen des Brüsselators kann man dem Bild 17.1 entnehmen. Die Sägezahnkurve stellt die Konzentration von Reagenz Y, die Stufenfunktion die Konzentration von Reagenz E dar. Die »Nadelspitzen«-Kurve zeigt die Konzentrationsänderung der Reagenz X, die für den Farbumschlag verantwortlich ist.

Vernachlässigt man den Einfluß der Reagenzien A, B; C; und E, so entsteht das vereinfachte System des Brüsselators

$$\dot{x} = a - (b+1)x + x^2 y$$
$$\dot{y} = bx - x^2 y$$

wobei die Konstante k_4 auf Eins normiert wurde. Der vereinfachte Brüsselator ist ein bekanntes System mit einem Grenzzyklus, den man z.B. für die Parameterwerte $a=0.4, b=1.7$ erhält.

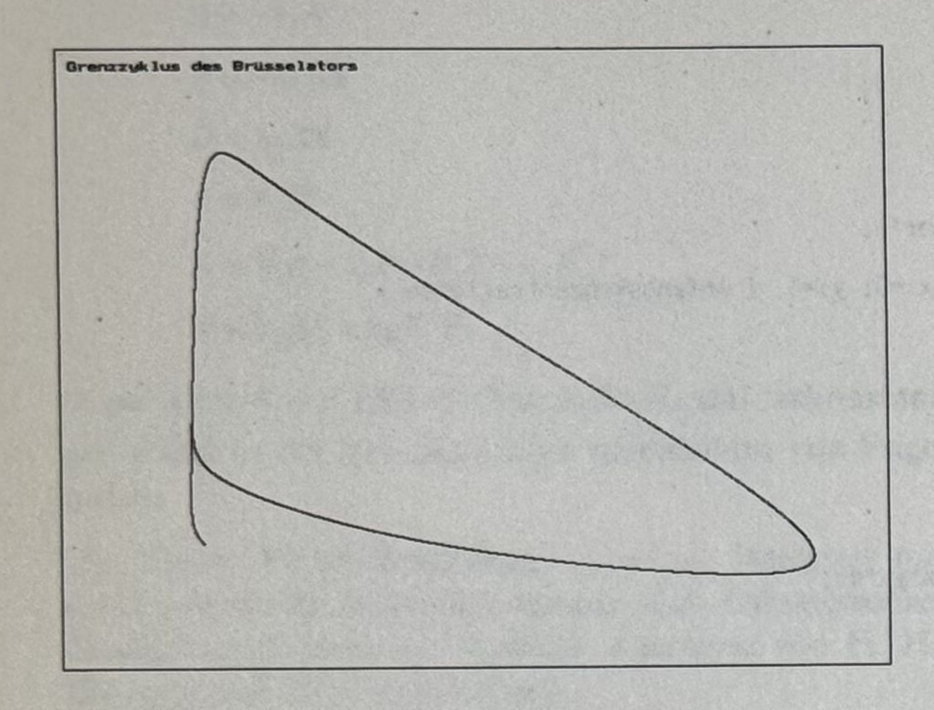


Bild 17.2 Grenzzyklus des vereinfachten Brüsselators

Der sinusförmig erregte Brüsselator wurde von K. Tomita und T. Kai untersucht. Dieses System ist im Abschnitt 7.8 behandelt worden.

17.2 Astronomie: Saturnringe

Die KAM-Theorie von Kap. 8 findet zahlreiche Anwendungen in der Astronomie. Insbesondere nimmt man an, daß die Materieverteilung des Saturnrings den »invarianten Tori« entspricht. Auch die Lücken im Asteriodengürtel werden durch die KAM-Theorie erklärt. Von J. Frφland stammt folgendes Modell zur Simulation der Materieverteilung in den Saturnringen. Der Radiusvektor und der Polarwinkel der Ringmaterie erfüllt die Rekursions-Gleichungen:

$$r_{n+1} = 2r_n - b_n + \frac{A\cos(\theta_n)}{(r_0 - r_n)^2}$$

$$b_{n+1} = r_n$$

$$\theta_{n+1} = \theta_n + \frac{2\pi r_0^{3/2}}{r_n^{3/2}}$$

Dabei ist A eine Konstante, die proportional zur Masse des innersten Saturn-Monds Mimas ist. Die Hilfsvariable h_{n+1} dient als Ersatz für den Abstand r_n, da das Rekursionsschema zweifach zurückgreifend ist. Der Anfangsabstand wird auf den Wert $r_0 = 1.857 \cdot 10^5\,km$ gesetzt; dies entspricht genau dem mittleren Abstand von Mimas, dem innersten Mond des Saturn.

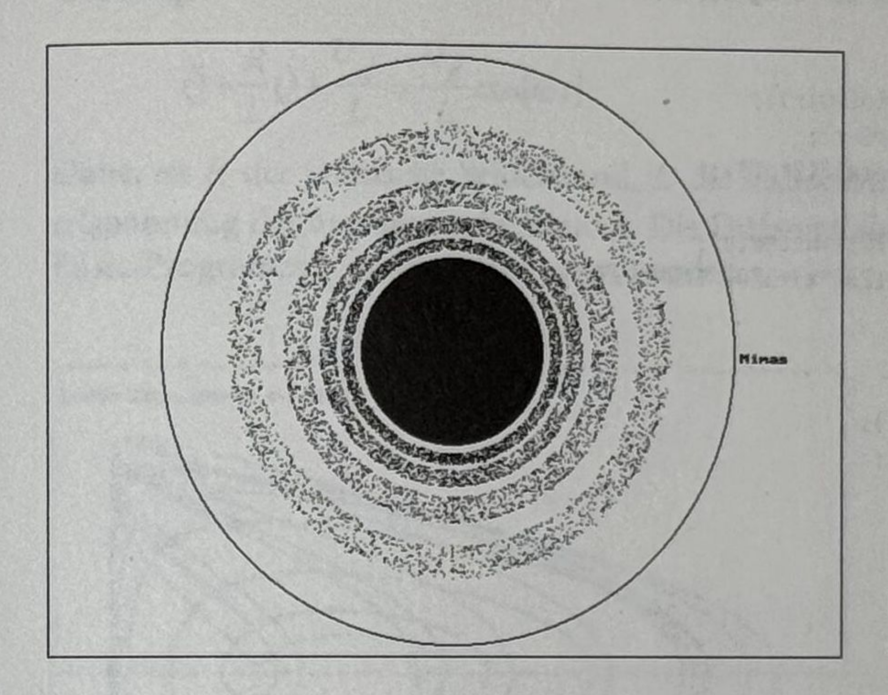


Bild 17.3 Saturnringe nach dem Modell von Frφland

Eine Realisierung des Modells von Frφland bietet das Turbo Pascal-Programm `saturn.pas`. Bild 17.3 zeigt den Graphik-Ausdruck des Programms. Es werden Anfangswerte für vier Ringe gegeben. Die Lücken zwischen den Ringen sind (von außen nach innen) die 1:4, 1:3 und 1:2 Lücke.

```
program saturn;
{ Modell für die Entstehung der Saturnringe
  nach J.Frφland, Introduction to Chaos and Coherence
  Institute of Physics, Bristol 1992 }
{$N+}
uses crt,graph;

const r0 = 1.857e5; { Radius der Mimas-Bahn}
      A = 2.0e12;
      d : array[1..4] of real= (7.0e4,8.44e4,10.0e4,13.0e4);
var r,r1,theta,theta1,h,x,y:double;
    i,j : integer;
    Graphdriver,Graphmode:integer;

begin
GraphDriver := Detect;
Initgraph(GraphDriver,GraphMode,'\tp\bgi');
SetgraphMode(Graphmode);
```

```
circle(320,240,73); { Saturn }
circle(320,240,230); { Mimasbahn }

for j:=1 to 4 do { Anzahl der Ringe }
begin
r := d[j];
h := r; theta := 0;
for i:=1 to 3000 do  { Verteilung der Ringmaterie }
   if r>0 then
   begin
   theta1 := theta+2*pi*exp(1.5*ln(r0/r));
   r1 := 2*r-h+A*cos(theta)/sqr(r0-r);
   theta := theta1-2*pi*trunc(theta1/(2*pi));
   h := r; r := r1;
   x := r/r0*cos(theta); y := r/r0*sin(theta);
   if i>200 then putpixel (trunc(230*x)+320, trunc(230*y)  +240 ,j);
   end
end;
floodfill(320,240,15);
moveto(300,320);outtext('Saturn');
moveto(555,240);outtext('Mimas');
repeat until keypressed;
closegraph;
textmode(lastmode)
end.
```

17.3 Elektronik: Dioden-Schwingkreis

Betrachtet werde ein Schwingkreis mit Diode nach Bild 17.4.

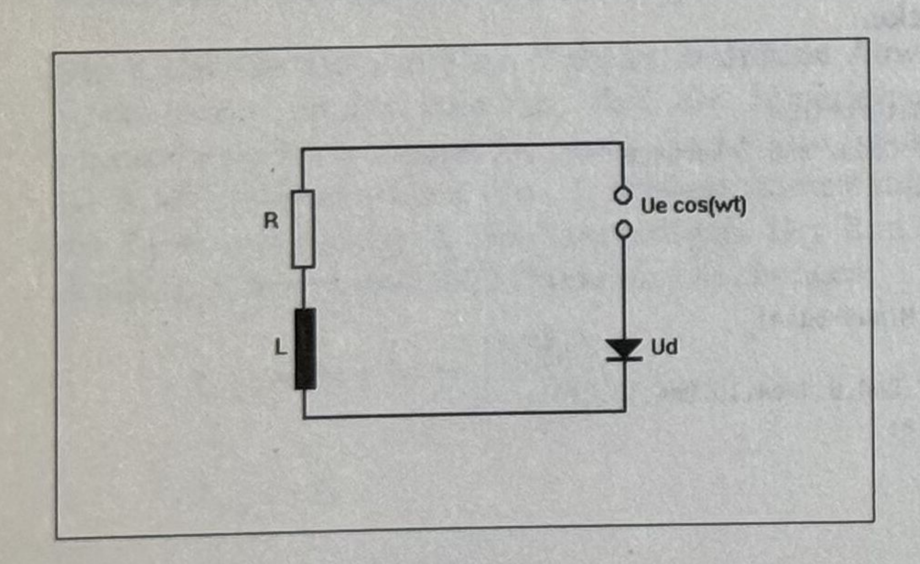


Bild 17.4 Elektromagnetischer Schwingkreis mit Diode

Approximiert man die Kennlinie der Diode durch eine Exponentialfunktion, so läßt sich die anliegende Spannung beschreiben durch

$$U_d = U_0\left(e^{\frac{Q}{C_0 U_0}} - 1\right)$$

Dabei kennzeichnen die Materialkonstanten C_0, U_0 den Typ der Diode. Für die durch den Stromkreis fließende Ladung Q ergibt sich die Differentialgleichung zweiter Ordnung

$$\ddot{Q} + \frac{R}{L}\dot{Q} + \frac{U_d}{L} = \frac{U_e}{L}\cos(\omega t)$$

Dabei ist R der Ohmsche Widerstand, L die Induktivität der Spule und U_e die Scheitelspannung der angelegten Spannung. Die Differentialgleichung kann mit dem Quick Basic-Programm `schwing.bas` integriert werden.

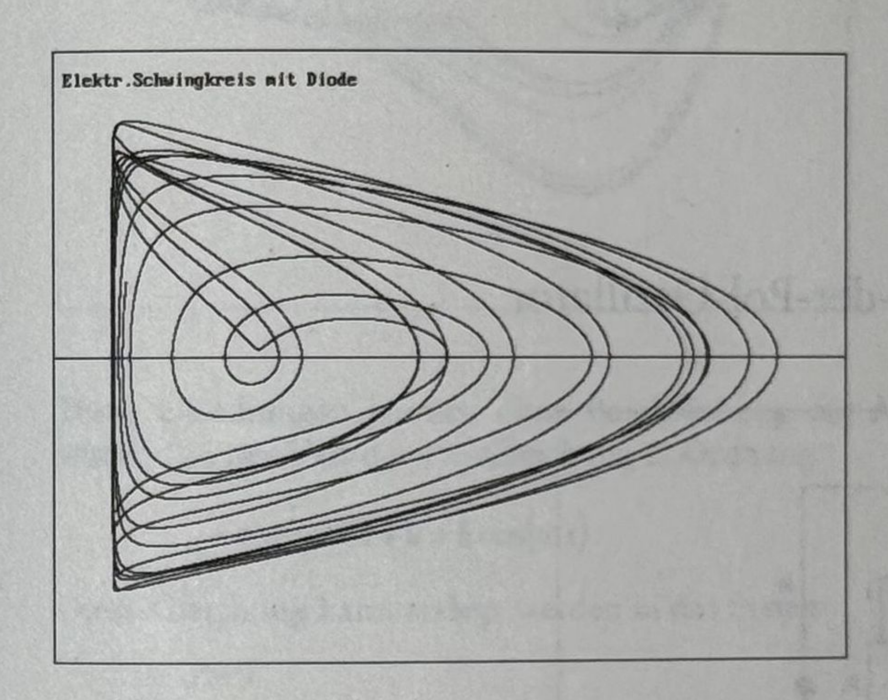


Bild 17.5 (I,U)-Phasenplot bei $U_e = 1.7V$

Im Programm wurden folgende Werte gewählt:

$$U_0 = 1.2V;\ C_0 = 0.025F;\ R = 1\Omega;\ L = 1H;\ \omega = 2\pi\frac{1}{s}$$

Es zeigt sich bei diesen Werten, daß mit wachsender äußerer Spannung U_e Periodenverdoppelungen auftreten (ab $1.6V$), die schließlich bei $2.07V$ zum Chaos führen.

```
'schwing.bas
'Elektrischer Schwingkreis mit Diode

CONST L = 1!:       'Induktivität in Henry
CONST R = 1:        'Ohmscher Widerstand
CONST U0 = 1.2:     'charakt.Diodenspannung
CONST C0 = .025:    'Kapazität der Diode bei U=0
```

```
CONST Ue = 2.2:   'externe Scheitelspannung
CONST dt = .0001: 'Zeitschritt in Sekunden
CONST PI = 3.141592653#

DIM Q, Qp, Qpp, U AS DOUBLE
SCREEN 12: CLS
WINDOW (-2, -1)-(9, 1)
LINE (-2, -1)-(9, 1), 15, B
LINE (-2, 0)-(9, 0), 15
LOCATE 2, 2: PRINT "Elektr.Schwingkreis mit Diode"

t = 0: Q = 0: Qp = 0: 'Anfangswerte
DO
  U = U0 * (EXP(Q / (U0 * C0)) - 1): 'Diodenspannung
  Qpp = -Qp * R / L - U / L + Ue / L * COS(2 * PI * t)
  Qp = Qp + Qpp * dt
  Q = Q + Qp * dt
  IF t > 3 THEN PSET (U, Qp), 14
  t = t + dt
LOOP UNTIL t > 20
e$ = INPUT$(1): SCREEN 0
END
```

17.4 Elektronik: Van-der-Pol-Oszillator

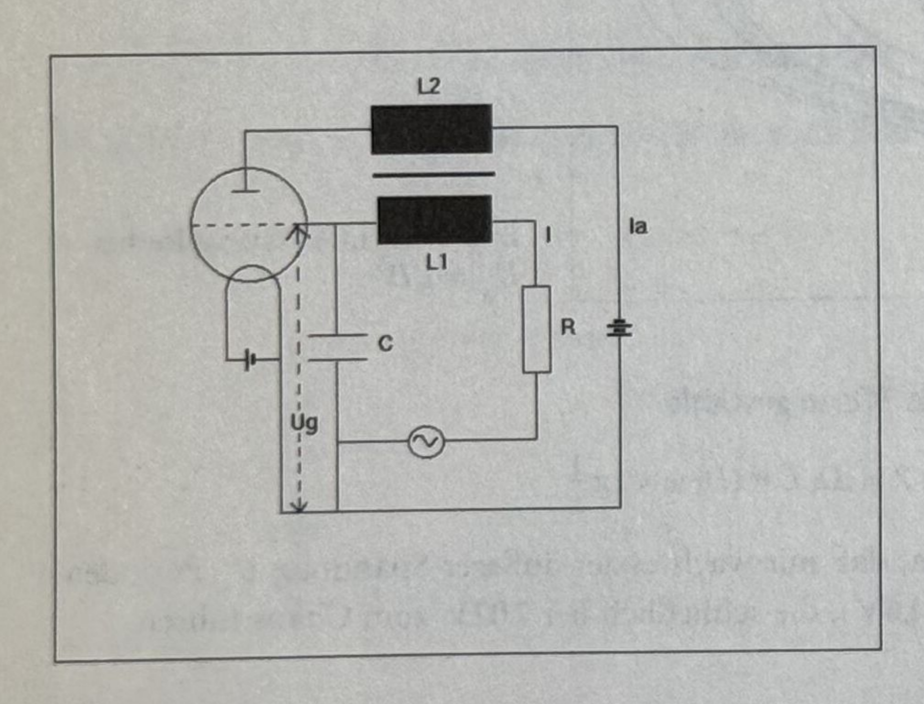


Bild 17.6 Schaltplan des van-der-Pol-Oszillators

Der Oszillator wurde 1926 von dem Holländer Balthasar van der Pol entdeckt und eingehend untersucht. Der Schaltkreis wird beschrieben durch die Gleichungen

$$L_1\dot{I} + RI + U_g - L_2\dot{I}_a = U_e\cos(\omega t); \quad C\dot{U}_g = 1$$

$$I_a = \alpha U_g\left(1 - \frac{U_g^2}{3\beta^2}\right)$$

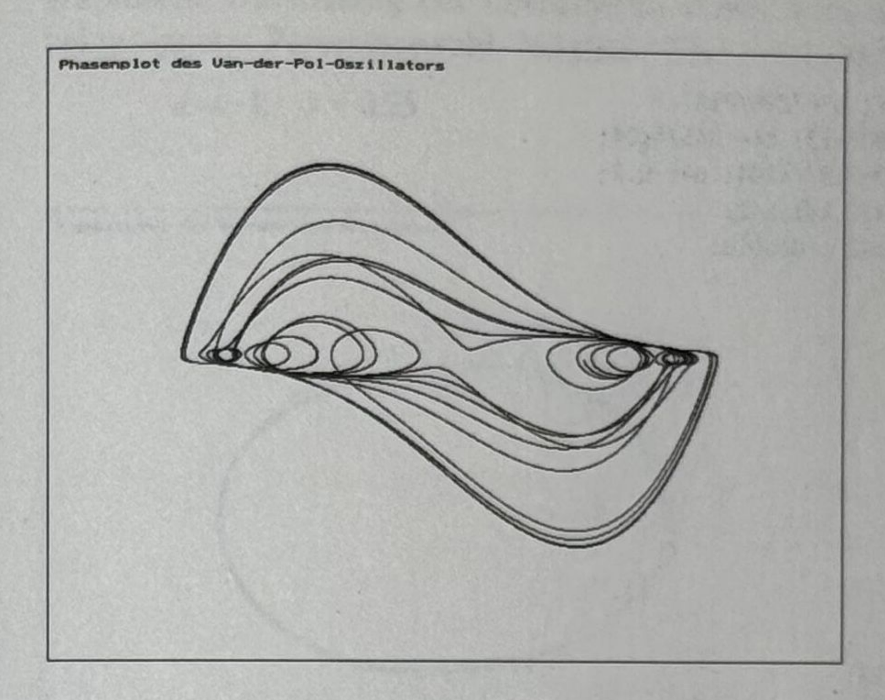


Bild 17.7 Phasenplot des Van-der-Pol-Oszillators ($a = 5, k = 5$)

Diese Gleichungen können ohne Beschränkung der Allgemeinheit zusammengefaßt werden zu der Van-der-Pol-Gleichung 2. Ordnung

$$\ddot{u} + a(u^2 - 1)\dot{u} + u = k\cos(\omega t)$$

Diese Gleichung kann zerlegt werden in das System

$$\dot{y} = x$$

$$\dot{x} = -a(y^2 - 1)x - y + k\cos(\omega t)$$

Das System wird integriert durch das Turbo Pascal-Programm `vandpol.pas`.

```
program vandpol;
{$N+}
uses crt,graph;

const omega = 2.4667; a = 5.0; k = 5.0;
      h = 0.001;
var y1,y2,t : double;

    Graphdriver,Graphmode:integer;

function f1(y1,y2:double):double;
begin
```

```
    f1 := y2
end;
function f2(y1,y2:double):double;
begin
    f2 := a*(1-y1*y1)*y2-y1+k*cos(omega*t)
end;

procedure rk(var y1,y2:double);
const a1=3/32; a2=9/32;
      b1=1932/2197; b2=-7200/2197; b3=7296/2197;
      c1=439/216; c2=-8.0; c3=3680/513; c4=-845/4104;
      d1=25/216; d2=1408/2565; d3=2197/4104; d4=-0.2;
var k01,k02,k11,k12,k21,k22,k31,k32,k41,k42,
    u11,u12,u21,u22,u31,u32,u41,u42 : double;
begin
k01 := f1(y1,y2);
k02 := f2(y1,y2);
u11 := y1+h*k01/4;
u12 := y2+h*k02/4;
k11 := f1(u11,u12);
k12 := f2(u11,u12);
u21 := y1+h*(a1*k01+a2*k11);
u22 := y2+h*(a1*k02+a2*k12);
k21 := f1(u21,u22);
k22 := f2(u21,u22);
u31 := y1+h*(b1*k01+b2*k11+b3*k21);
u32 := y2+h*(b1*k02+b2*k12+b3*k22);
k31 := f1(u31,u32);
k32 := f2(u31,u32);
u41 := y1+h*(c1*k01+c2*k11+c3*k21+c4*k31);
u42 := y2+h*(c1*k02+c2*k12+c3*k22+c4*k32);
k41 := f1(u41,u42);
k42 := f2(u41,u42);
y1 := y1+h*(d1*k01+d2*k21+d3*k31+d4*k41);
y2 := y2+h*(d1*k02+d2*k22+d3*k32+d4*k42);
end;

begin
GraphDriver := Detect;
Initgraph(GraphDriver,GraphMode,'\tp\bgi');
SetgraphMode(Graphmode);
rectangle(0,0,639,479);
moveto(10,10);
outtext('Phasenplot des Van-der-Pol-Oszillators');

y1 := 0.8; y2 := 0.6; t := 0;
while t<= 300 do
  begin
  rk(y1,y2);
  t := t+h;
  if t>10 then putpixel(320+round(100*y1),240+round(15*y2), 1+trunc(t/23));
  end;
```

```
repeat until keypressed;
closegraph;
textmode(lastmode)
end.
```

Der Van-der-Pol-Oszillator ist ein bekanntes System mit einem Grenzzyklus. Eine graphische Darstellung des Grenzzyklus erhält man mit dem gegebenen Programm bei geeigneter Parameterwahl. Mögliche Parameter sind z. B.

$$a = -1; \quad k = 0.55$$

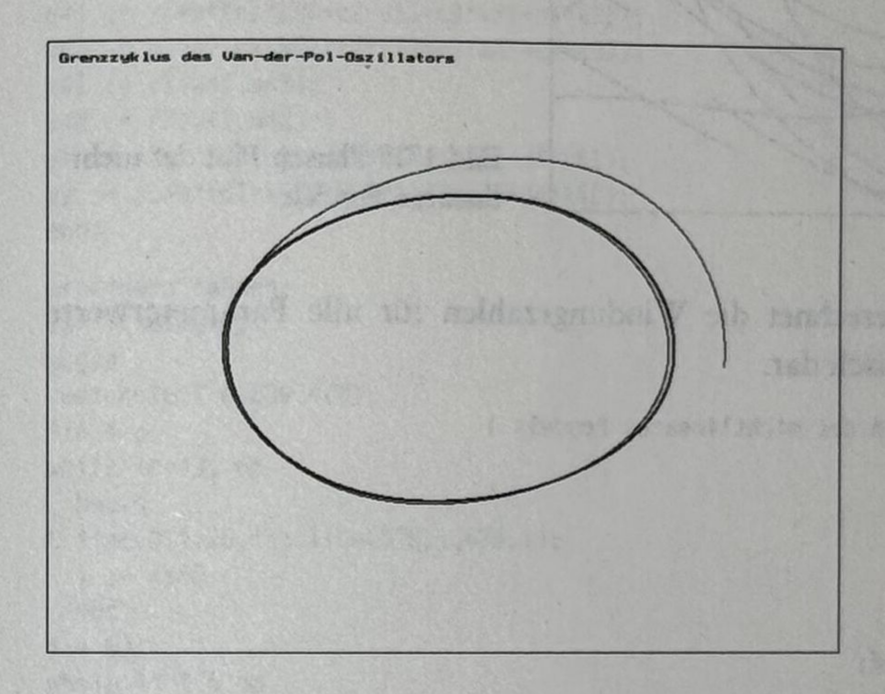


Bild 17.8 Grenzzyklus des Van-der-Pol-Oszillators

17.5 Physik: Nichtlineares Pendel

Für das sinusförmig erregte Fadenpendel läßt sich folgende (nichtlineare) Gleichung aufstellen

$$ml\ddot{\varphi} + \gamma\dot{\varphi} + \sin\varphi = A\cos(\omega t)$$

Diese Gleichung kann ohne Beschränkung der Allgemeinheit vereinfacht werden zu

$$\ddot{\varphi} + \frac{1}{q}\dot{\varphi} + \sin\varphi = g\cos(\omega t)$$

Letztere Gleichung ist äquivalent zum System der Differentialgleichungen

$$\dot{y} = -\frac{1}{q}y - \sin x + g\cos(\omega t)$$

$$\dot{x} = y$$

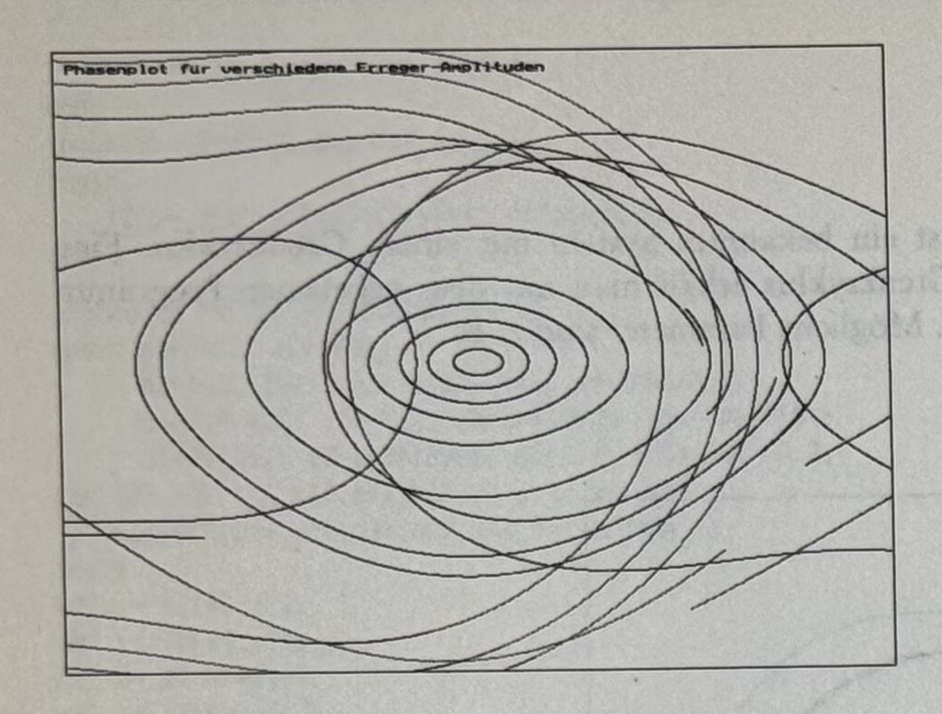


Bild 17.9 Phasen-Plot des nichtlinearen Pendels

Das Programm pendel2.pas berechnet die Windungszahlen für alle Parameterwerte $a \in [1;1.55]$ und stellt sie graphisch dar.

```
program pendel2; { Windungszahlen des nichtlinearen Pendels }
{$N+}
uses crt,graph;

const omega = 0.6667; q = 2.0;
      h = 0.05;
var y01,y1,y2,t,g,w : double;
    Graphdriver,Graphmode:integer;

function f1(y1,y2:double):double;
begin
    f1 := y2
end;
function f2(y1,y2:double):double;
begin
    f2 := -y2/q-sin(y1)+g*cos(omega*t)
end;

procedure rk(var y1,y2:double);
const a1=3/32; a2=9/32;
      b1=1932/2197; b2=-7200/2197; b3=7296/2197;
      c1=439/216; c2=-8.0; c3=3680/513; c4=-845/4104;
      d1=25/216; d2=1408/2565; d3=2197/4104; d4=-0.2;
var k01,k02,k11,k12,k21,k22,k31,k32,k41,k42,
    u11,u12,u21,u22,u31,u32,u41,u42 : double;
begin
k01 := f1(y1,y2);
k02 := f2(y1,y2);
u11 := y1+h*k01/4;
```

```
ul2 := y2+h*k02/4;
k11 := f1(u11,u12);
k12 := f2(u11,u12);
u21 := y1+h*(a1*k01+a2*k11);
u22 := y2+h*(a1*k02+a2*k12);
k21 := f1(u21,u22);
k22 := f2(u21,u22);
u31 := y1+h*(b1*k01+b2*k11+b3*k21);
u32 := y2+h*(b1*k02+b2*k12+b3*k22);
k31 := f1(u31,u32);
k32 := f2(u31,u32);
u41 := y1+h*(c1*k01+c2*k11+c3*k21+c4*k31);
u42 := y2+h*(c1*k02+c2*k12+c3*k22+c4*k32);
k41 := f1(u41,u42);
k42 := f2(u41,u42);
y1 := y1+h*(d1*k01+d2*k21+d3*k31+d4*k41);
y2 := y2+h*(d1*k02+d2*k22+d3*k32+d4*k42);
end;

procedure rahmen;
var i:integer;
begin
rectangle(0,0,639,479);
i:= 47;
while i<=431 do
  begin
  line(0,i,10,i); line(629,i,639,i);
  i := i+48;
  end;
i:= 63;
while i<=575 do
  begin
  line(i,0,i,10); line(i,469,i,479);
  i := i+64;
  end;
end;

begin
GraphDriver := Detect;
Initgraph(GraphDriver,GraphMode,'\tp\bgi');
SetgraphMode(Graphmode);
rahmen;
moveto(10,10);outtext('Windungsdiagramm des Pendels');

g := 1.0; y01 := 0.1;
while g<= 1.55 do
begin
y1 := y01; y2 := 0.1; t := 0;
while t<= 250 do
  begin
  rk(y1,y2);
  t := t+h;
```

```
  end;
w := (yl-y01)/(250/h);
putpixel(round((g-1)*1163),240+round(2000*w),14);
putpixel(round((g-1)*1163),240-round(2000*w),14);
g := g+0.0008594;
end;
repeat until keypressed;
closegraph;
textmode(lastmode)
end.
```

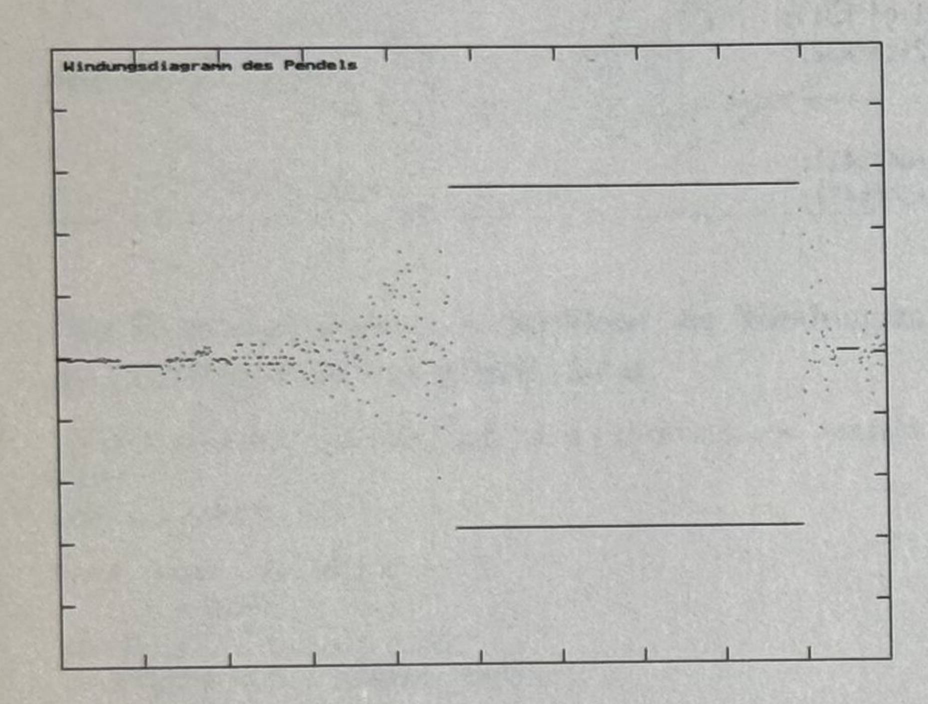


Bild 17.10 Windungs-Diagramm des nicht-linearen Pendels

Die genaue Analyse der Graphik zeigt folgendes Verhalten in Abhängigkeit vom Parameter $g \in [1;1.55]$.

Intervall	Verhalten	Windungszahl
$g < 1.14$	periodisch	0
$1.14 < g < 1.28$	chaotisch mit 1 period.Fenster	unbestimmt
$1.28 < g < 1.475$	periodisch	± 1
$1.475 < g < 1.493$	periodisch mit 2 chaot.Fenstern	± 1
$1.493 < g < 1.495$	chaotisch	unbestimmt
$1.495 < g < 1.497$	periodisch	$\pm \frac{2}{3}$
$g > 1.497$	chaotisch	unbestimmt

Sehr viel aufwendiger ist die Bestimmung eines Poincaré-Schnitts. Da die Pendelgleichung unter der folgenden Transformation invariant ist, wird hier gewählt

$$t \rightarrow t + \frac{2\pi n}{\omega}$$

Das Poincaré-Diagramm kann mit Hilfe des Turbo Pascal-Programm `pendel3.pas` erstellt werden.

```
program pendel3; { Poincaré-Diagramm des nichtlinearen Pendels }
{$N+}
uses crt,graph;

const omega = 0.66667; q = 4.0; g = 1.5;
      h = 0.01;
var y1,y2,t : double;
    Graphdriver,Graphmode:integer;
    n:longint;

function f1(y1,y2:double):double;
begin
    f1 := y2
end;
function f2(y1,y2:double):double;
begin
    f2 := -y2/q-sin(y1)+g*cos(omega*t)
end;

procedure rk(var y1,y2:double);
const a1=3/32; a2=9/32;
      b1=1932/2197; b2=-7200/2197; b3=7296/2197;
      c1=439/216; c2=-8.0; c3=3680/513; c4=-845/4104;
      d1=25/216; d2=1408/2565; d3=2197/4104; d4=-0.2;
var k01,k02,k11,k12,k21,k22,k31,k32,k41,k42,
    u11,u12,u21,u22,u31,u32,u41,u42 : double;
begin
k01 := f1(y1,y2);
k02 := f2(y1,y2);
u11 := y1+h*k01/4;
u12 := y2+h*k02/4;
k11 := f1(u11,u12);
k12 := f2(u11,u12);
u21 := y1+h*(a1*k01+a2*k11);
u22 := y2+h*(a1*k02+a2*k12);
k21 := f1(u21,u22);
k22 := f2(u21,u22);
u31 := y1+h*(b1*k01+b2*k11+b3*k21);
u32 := y2+h*(b1*k02+b2*k12+b3*k22);
k31 := f1(u31,u32);
k32 := f2(u31,u32);
u41 := y1+h*(c1*k01+c2*k11+c3*k21+c4*k31);
u42 := y2+h*(c1*k02+c2*k12+c3*k22+c4*k32);
k41 := f1(u41,u42);
k42 := f2(u41,u42);
y1 := y1+h*(d1*k01+d2*k21+d3*k31+d4*k41);
```

```
y2 := y2+h*(d1*k02+d2*k22+d3*k32+d4*k42);
end;

begin
GraphDriver := Detect;
Initgraph(GraphDriver,GraphMode,'\tp\bgi ");
SetgraphMode(Graphmode);
rectangle(0,0,639,479);
moveto(10,10);outtext( "Poincare-Schnitt;q=4;g=1.5 ");

y1 := 0.8; y2 := 0.6; t := 0; n := 0;
while t<= 100000 do
  begin
  rk(y1,y2);
  t := t+h;
  y1 := y1-2*pi*abs(y1)/y1;
  if (n>10) and (abs(t-2*pi*n/omega)<0.01) then
      putpixel(320+round(100*y1),340-round(100*y2),14);
  if (t>2*pi*n/omega) then n := n+1;
  end;
repeat until keypressed;
closegraph;
textmode(lastmode)
end.
```

Einen einfacheren Zugang zur Pendel-Gleichung bietet das diskretisierte Newton-Verfahren des nächsten Abschnitts.

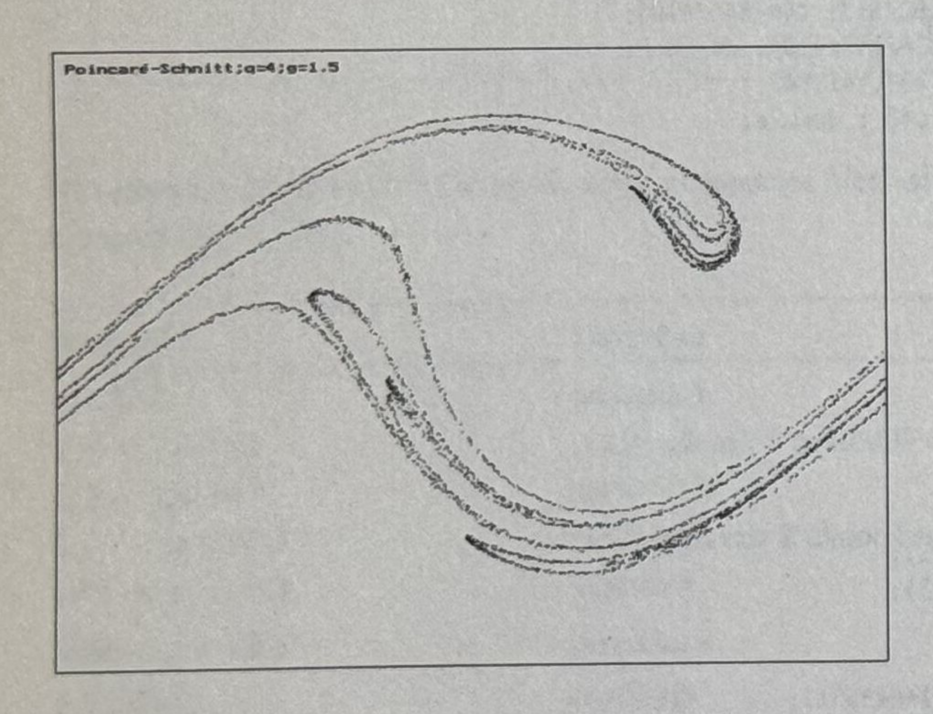


Bild 17.11 Poincaré-Schnitt des nichtlinearen Pendels

17.6 Physik: Diskretes Newton-Verfahren

Setzt man beim Zeitschritt h die Bewegungsgleichungen

$$v_{n+1} = \frac{x_{n+1} - x_n}{h}; \quad v_n = \frac{x_n - x_{n-1}}{h}; v_{n+1} = v_n + ah$$

zusammen, so erhält man die zweifach zurückgreifende Beziehung

$$x_{n+1} - x_n = x_n - x_{n-1} + ah^2$$

Dies läßt sich als rekursives Schema schreiben

$$x_{n+1} = 2x_n - y_n - ahx_n^2; \quad y_{n+1} = x_n$$

Für das Fadenpendel ergibt sich schließlich das diskretisierte Newton-Verfahren

$$x_{n+1} = 2x_n - y_n - \frac{g}{l}h^2 \sin x_n; \quad y_{n+1} = x_n$$

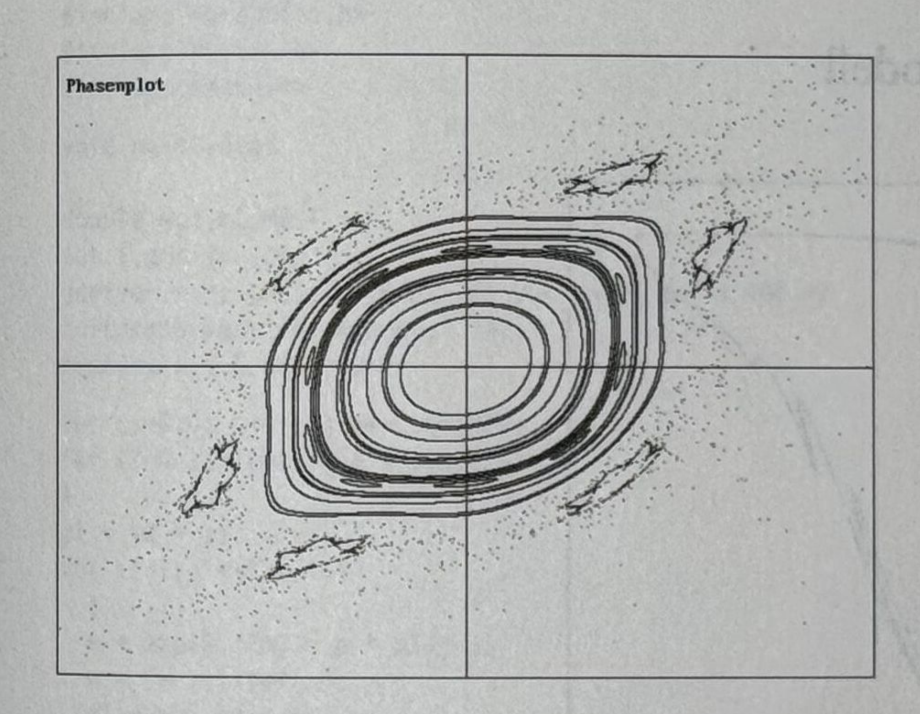


Bild 17.12 Phasenplot bei verschiedenen Anfangswerten

Das Verfahren ist im Quick Basic-Programm `disnewt.bas` implementiert. Das Produkt der Konstanten $\frac{g}{l}h^2$ ist im Programm als Parameter a einzugeben.

```
"disnewt.pas
"Diskretisierung des Newton-Kraftgesetzes

CONST a = 1.71
DIM x, x1, y AS DOUBLE
DIM i, j AS INTEGER

SCREEN 12: CLS
```

```
WINDOW (-3.2, -3.2)-(3.2, 3.2)
LINE (-3.2, -3.2)-(3.2, 3.2), , B
LINE (-3.2, 0)-(3.2, 0)
LINE (0, -3.2)-(0, 3.2)
LOCATE 2, 2: PRINT "Phasenplot"

RANDOMIZE TIMER
FOR i = 1 TO 40
x = 3 * RND - 3: y = 3 * RND - 3
FOR j = 1 TO 20000
  x1 = 2 * x - y - a * SIN(x)
  y = x
  x = x1
  PSET (x, y), 1 + i \ 3
NEXT j
NEXT i
e$ = INPUT$(1): SCREEN 0
END
```

17.7 Physik: Ising-Modell

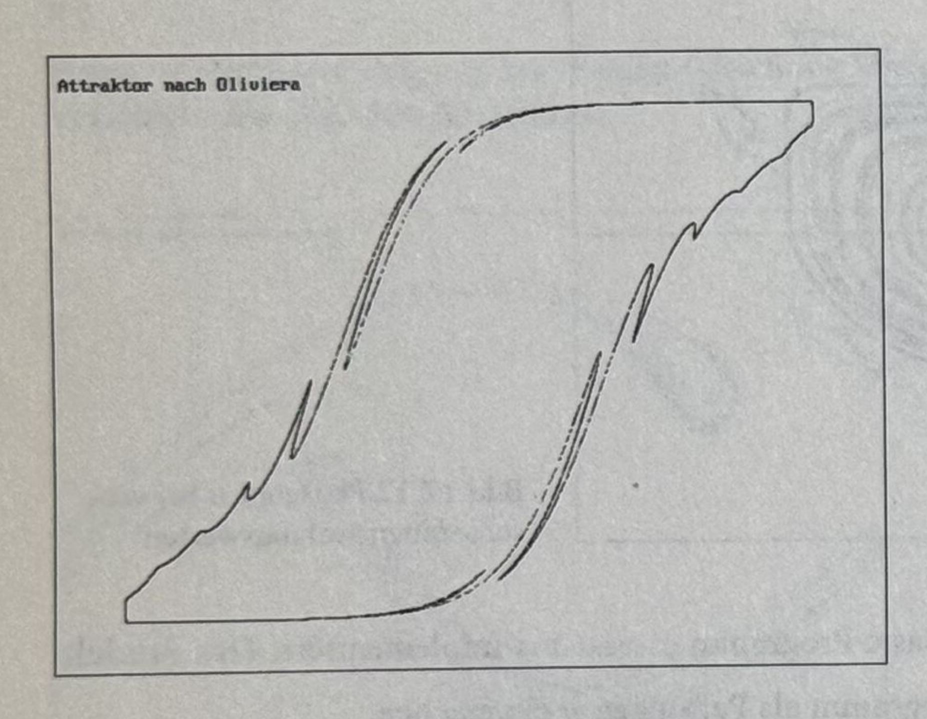


Bild 17.13 Attraktor nach Oliviera

Für die Magnetisierung nach dem Ising-Modell konnten M. J. de Oliviera u. a. (*Devil´s Staircase and Strange Attractor in the Ising Modell* [24], Seite 427) folgendes Modell aufstellen

$$M_{n+1} = \tanh \frac{M_n - pM_{n-1} + H}{T}$$

Dabei ist n die Anzahl der berücksichtigten Spin-Zonen, p, T sind Parameter des Modells. Oliviera fand durch numerische Berechnung der Ljapunow-Exponenten ($\lambda = 0.12$) Evidenz für einen seltsamen Attraktor bei den Parameterwerten $T = 0.2; \quad p = 0.69662$

Der (vermutliche) Attraktor hat das Aussehen einer Hysteresis-Schleife.

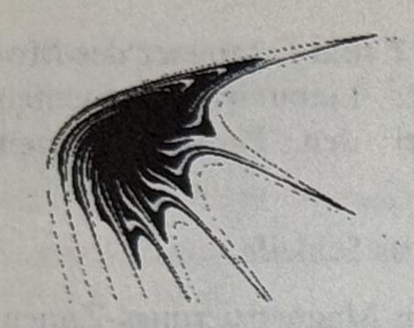
Das Bild im Phasenraum von (T, p) liefert die verschiedenen Magnetisierungs-Zonen (siehe Farbtafel 3). Die linke obere Ecke stellt die Zone des Paramagnetismus dar; der unterste Eckpunkt heißt Lifschitz-Punkt. Die linke, untere Ecke bildet den Bereich des Ferromagnetismus. Der blütenförmige Rest der Abbildung bildet die modulierte magnetische Phase.

Der Phasenraum im Bereich $p \times T = [0.4;1.6] \times [0.1;1]$ kann mit dem Turbo C-Programm `ising2.c` dargestellt werden.

```
/* ising2.c */
#include <stdio.h>
#include <graphics.h>
#include <conio.h>
#include <math.h>

void main(void)
{
double e,m,m1,m2,T,p;
int i,gdriver,gmode;
gdriver = 9; gmode = VGAHI; /* VGA-Modus 640 x 480 */
initgraph(&gdriver,&gmode," ");
rectangle(0,0,639,479);

for (p=0.4; p<=1.6; p+=0.001875)
for (T=0.1; T<=1.0; T+=0.001875)
{
m1 = m2 = 1;
for (i=1; i<=50; i++)
  {
  e = exp(2.*(m2 - p * m1)/T);
  m = (e-1.)/(e+1.);
  m1 = m2; m2 = m;
  }
putpixel((int)((p-0.4)*533.),480-(int)((T-0.1)*533.),1+(int)((m+1)*7));
}
do {} while(!(kbhit()));
closegraph();
return;
}
```



18 Anwendungen II

In diesem Abschnitt werden weitere Anwendungen aus der Biologie, der Theorie der Zellularautomaten und Clusterbildung vorgestellt.

18.1 Biologie: Lotka-Volterra-Gleichungen

Das bekannteste Modell eines Räuber-Beute-Systems stammt von Volterra und Lotka. Mit Hilfe dieses Modells konnte die dramatische Abnahme des Sardinen-Fischfangs (nach dem ersten Weltkrieg) erklärt werden. Ist x die Beute- und y die Räuber-Population, so gelten die Lotka-Volterra-Gleichungen

$$\dot{x} = ax - bxy = f(x,y)$$
$$\dot{y} = -cy + dxy = g(x,y)$$

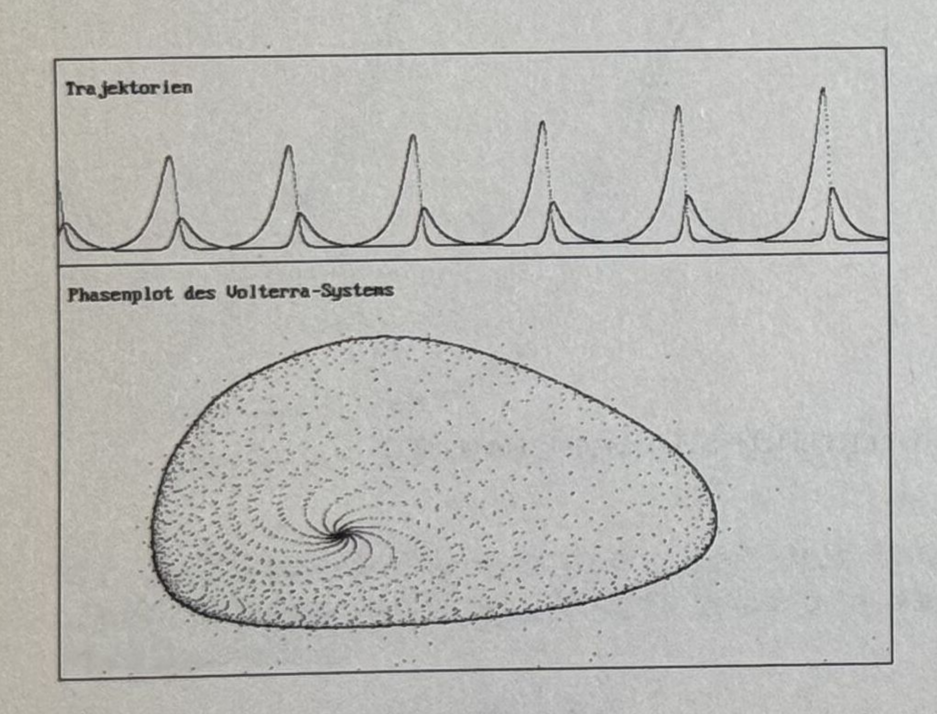


Bild 18.1 Trajektorien und Phasenplot des Volterra-Modells

dabei sind a,b,c,d positive Konstanten. a,c sind die Geburtenrate der Beute bzw. Sterberate des Räubers. b,d geben die Abnahme der Beute bzw. den Zuwachs des Räubers an aufgrund der Begegnungen Räuber-Beute. Der Fixpunkt ist ein stationärer Punkt des Systems; man erhält ihn bei verschwindenden zeitlichen Ableitungen. Diese Bedingung liefert den (nichttrivialen) Fixpunkt

$$(x_F|y_F)=\left(\frac{c}{d}\middle|\frac{a}{b}\right)$$

Für $x=0 \wedge y=0$ erhält man den Ursprung als trivialen Fixpunkt, der aber biologisch uninteressant ist. Ob oder wie schnell der nichttriviale Fixpunkt erreicht wird, hängt von der Dynamik des Systems ab.

Der obere Teil von Bild 18.1 liefert die typischen Trajektorien eines Räuber-Beute-Modells. Die Maxima der zahlenmäßig geringeren Räuber hinken zeitlich hinter den Maxima der zahlenmäßig überwiegenden Beutetiere nach. Im unteren Teil der Abbildung befindet sich ein Phasenplot für verschiedene Anfangsparameter. Deutlich sieht man hier, wie sich die Kurven entweder der Grenzkurve oder dem Fixpunkt nähern.

Dieses stetige Modell von Lotka-Volterra kann auf verschiedene Arten diskretisiert werden. Eine Mischung aus den Verfahren von Euler und Heun stellt der folgende Algorithmus dar.

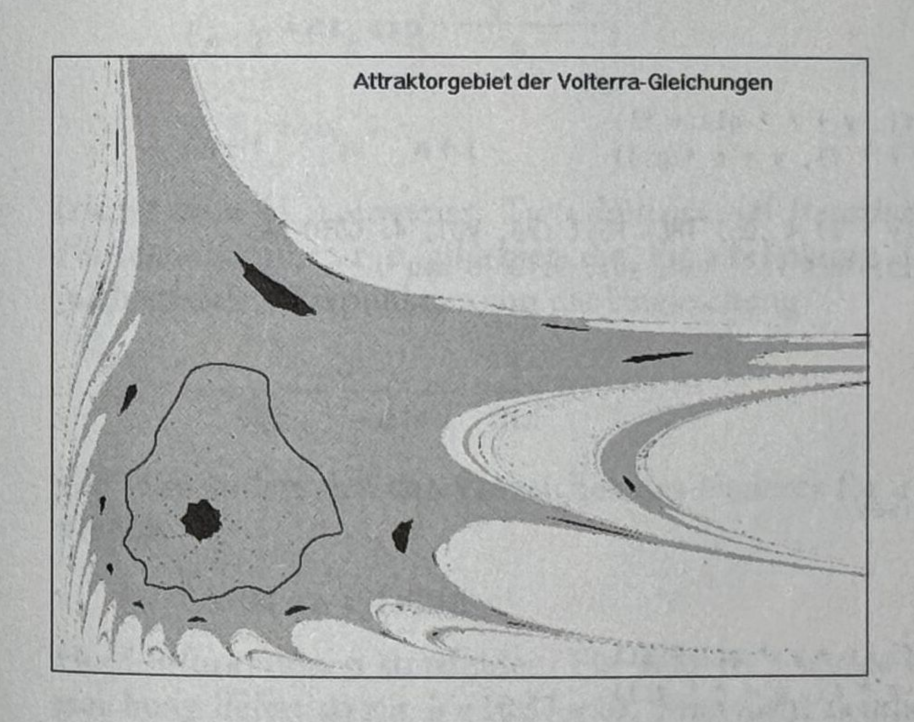


Bild 18.2 Attraktionsgebiet des Volterra-Systems

$$x_{n+1} = x_n + \tfrac{1}{2}h\big(f(x,y) + f(x+rf(x,y)),y+rg(x,y)\big)$$
$$y_{n+1} = y_n + \tfrac{1}{2}h\big(g(x,y) + g(x+rf(x,y)),y+rg(x,y)\big)$$

Für $r=0$ erhält man die Eulersche Methode, für $r=h$ diejenige von Heun.

Setzt man die Parameter des Volterra-Systems konstant gleich 1 und faßt r,h als Parameter auf, so erhält man den Parameterraum (r,h). Peitgen und Richter [22] konnten zeigen, daß sich dort mindestens ein seltsamer Attraktor befindet. Für $h=0.8 \wedge r=0.86$ findet im Parameterraum eine Hopf-Bifurkation statt und es erscheinen Arnold-Zungen. Für $r=h=0.739$ gelangt man in die Nähe des Attraktors und findet eine anziehende, periodische Bahn und einen invarianten Kreis (Bild 18.2).

Das Attraktionsgebiet kann mit Hilfe des Quick Basic-Programms `volt2.bas` gezeichnet werden.

```
'volt2.bas
'Attraktionsgebiet des Volterra-Systems

CONST r = .86: h = .8
DIM f1, g1, x, y, x0, y0 AS DOUBLE
DEF fnf (x, y) = x - x * y
DEF fng (x, y) = -y + x * y

SCREEN 12: CLS
WINDOW (.01, 0)-(5.31, 4.01)
LINE (.01, 0)-(5.31, 4.01), , B

FOR x0 = .01 TO 5.31 STEP .008281
FOR y0 = .01 TO 4.01 STEP .008333
x = x0: y = y0
FOR i = 1 TO 100
  f1 = fnf(x, y): g1 = fng(x, y)
  x1 = x + h / 2 * (fnf(x + r * f1, y + r * g1) + f1)
  y1 = y + h / 2 * (g1 + fng(x + r * f1, y + r * g1))
  x = x1: y = y1
  IF (ABS(x - 1) < .05) AND (ABS(y - 1) < .05) THEN PSET (x0, y0), 4: GOTO neu
  IF (ABS(x) > 10) OR (ABS(y) > 10) THEN PSET (x0, y0), 9: GOTO neu
NEXT i
  PSET (x0, y0), 14
neu:
NEXT y0
NEXT x0

'Einzeichnen des invarianten Kreises
x = .999: y = .999
FOR i = 1 TO 3000
  f1 = fnf(x, y): g1 = fng(x, y)
  x1 = x + h / 2 * (fnf(x + r * f1, y + r * g1) + f1)
  y1 = y + h / 2 * (g1 + fng(x + r * f1, y + r * g1))
  x = x1: y = y1
  PSET (x, y), 12
NEXT i
e$ = INPUT$(1): SCREEN 0
END
```

18.2 Biologie: Modell von Metz

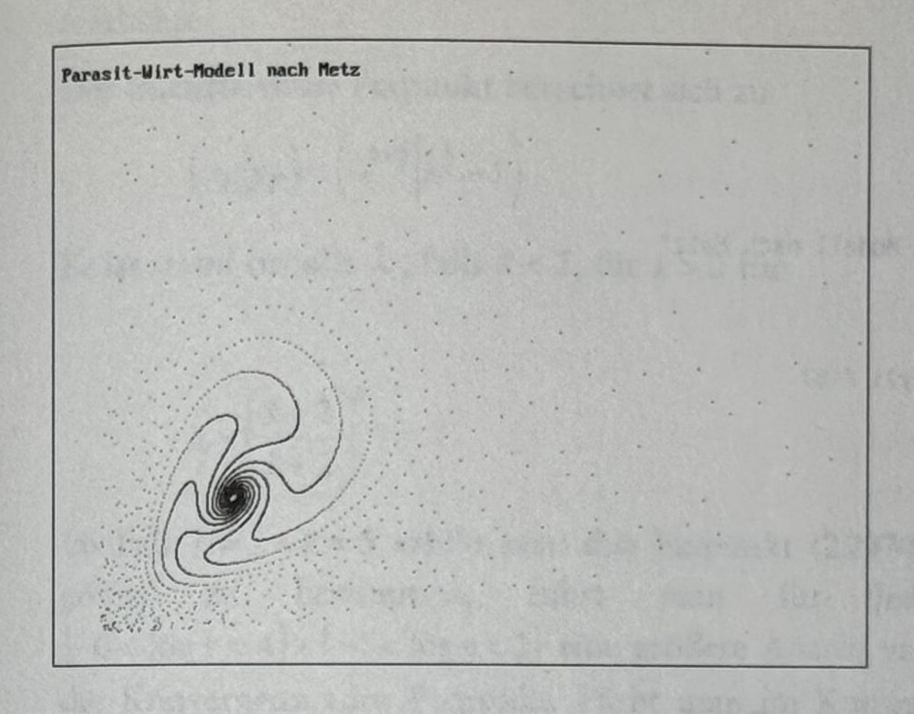


Bild 18.3 Hopf-Bifurkationen beim Modell von Metz

Von dem Biologen J. A. Metz stammt das folgende Wirt-Parasit-Modell

$$x_{n+1} = ax_n \exp\left(\frac{1-\sqrt{1+y_n}}{b}\right)$$

$$y_{n+1} = ax_n - x_{n+1}$$

(zitiert nach H. Lauwerier: *Two-dimensional Iterative Maps* im Sammelband [13]). Für den Parameter a gilt hier die Einschränkung $1 < a < 4.9216$. Es existiert ein (nichttrivialer) Fixpunkt, wenn die Ungleichung

$$b > \frac{a}{2(a-1) - a\ln a} - \frac{1}{\ln a}$$

gilt. Hier ändert sich das Vorzeichen des Nenners für $a = 1.6304$. Daraus folgt, daß im Bereich

$$1.6304 < a < 4.9216$$

Hopf-Bifurkationen stattfinden. Für Bild 18.3 wurde $a = 4.2$ gewählt, die obige Ungleichung liefert damit $b = 10.57 \approx 10$. Man sieht deutlich, daß der innere Zyklus anziehend, der äußere dagegen abstoßend ist. Die Abbildung wurde mit dem Basic-Programm `metz.bas` erstellt.

```
'metz.bas
'Parasit-Wirt-Modell nach J.Metz

CONST a = 4.145: b = 10
DIM i AS INTEGER
DIM x, x1, y AS DOUBLE
SCREEN 12: CLS
WINDOW (-20, -50)-(400, 1000)
LINE (-20, -50)-(400, 1000), , B
LOCATE 2, 2: PRINT "Parasit-Wirt-Modell nach Metz"

x = 200: y = 60
FOR i = 1 TO 5000
  x1 = a * x * EXP((1 - SQR(1 + y)) / b)
  y = a * x - x1
  x = x1
  PSET (x, y), 1 + i \ 380
NEXT i
e$ = INPUT$(1): SCREEN 0
END
```

18.3 Biologie : Modell von Crofton

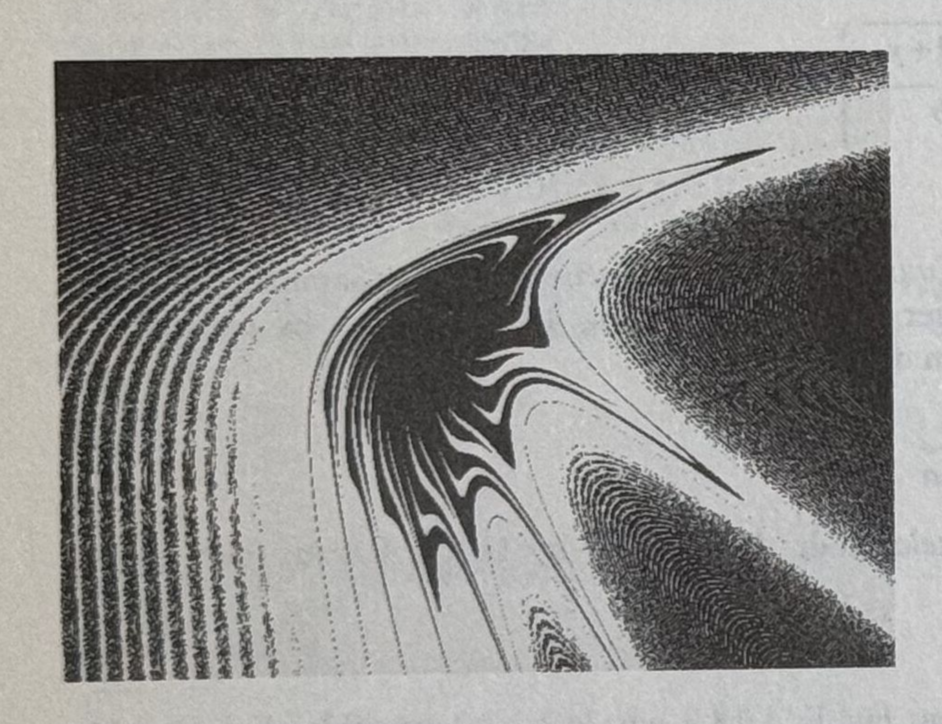


Bild 18.4 Attraktorgebiet des Crofton-Modells (Achsen logarithmisch)

Der Parasitologe H. D. Crofton publizierte 1971 das Wirt-Parasit-Modell

$$x_{n+1} = \frac{\lambda x_n}{(1+y_n)^k} \qquad y_{n+1} = \frac{x_n y_n}{(1+y_n)^{k+1}}$$

(Crofton H. D.: *A qualitative approach to parasitism*, Parasitology, 63). Der Parameter λ hat die Bedeutung der Wachstumsrate des Wirts, k ist ein Maß für die Parasitendichte.

Der (nichttriviale) Fixpunkt berechnet sich zu

$$\left(x_F \middle| y_F\right) = \left(\lambda^{1+\frac{1}{k}} \middle| \lambda^{\frac{1}{k}} - 1\right)$$

Er ist stabil für alle λ, falls $k < 2$, für $k > 2$ für

$$\lambda < \left(\frac{k-2}{k+2}\right)^k$$

Im Fall $\lambda = 2 \wedge k = 5$ erhält man den Fixpunkt $(2.2974|0.1487)$. Um das Attraktionsgebiet zu bestimmen, führt man für jeden Punkt des Bereichs $\{-6 < \log x < 4\} \times \{-4 < \log y < 2\}$ eine größere Anzahl von Iterationen durch und prüft die Konvergenz zum Fixpunkt. Färbt man im Konvergenzfall die Startpunkte entsprechend ein, so erhält man Bild 18.4. Das Vorgehen kann dem Pascal-Programm `crofton.pas` entnommen werden.

```
program crofton; { Wirt-Parasit-Modell nach Crofton }
{$N+}

uses crt,graph;
label naechst;
const lambda=2.0; k=5; eps = 5e-7;
      ln10= 2.302585093;
var s,x,x0,x1,xfix,y,y0,yfix : double;
    j,Graphdriver,Graphmode:integer;

begin
GraphDriver := Detect;
Initgraph(GraphDriver,GraphMode,'\tp\bgi');
SetgraphMode(Graphmode);

xfix := exp(1.2*ln(2)); yfix := exp(0.2*ln(2))-1;
x0 := -2.5;
while x0<=4.0 do
begin
  y0 := -3.5;
  while y0<=3.0 do
  begin
  x := exp(x0*ln10);
  y := exp(y0*ln10);
  for i:=1 to 200 do
     begin
     s := exp(-k*ln(1+y));
```

```
      xl := lambda*x*s;
      y := x*y*s/(1+y);
      x := xl;
      if y<1.0e-99 then
          begin
putpixel(round((x0+2.5)*98.4612),480-round((y0+3.5)* 98.4612),9);
          goto naechst;
          end;
      if (abs(x-xfix)<eps) and (abs(y-yfix)<eps) then
          begin
   putpixel(round((x0+2.5)*98.4612),480-round((y0+3.5)* 98.4612), 12);
          goto naechst;
          end;
          end;
  putpixel(round((x0+2.5)*98.4612),480-round((y0+3.5)* 98.4612),14);
  naechst:
  y0 := y0+0.0101562285;
  end;
x0 := x0+0.0101562285;
end;
repeat until keypressed;
closegraph;
textmode(lastmode)
end.
```

18.5 Zellularautomat mit 3 Nachbarn

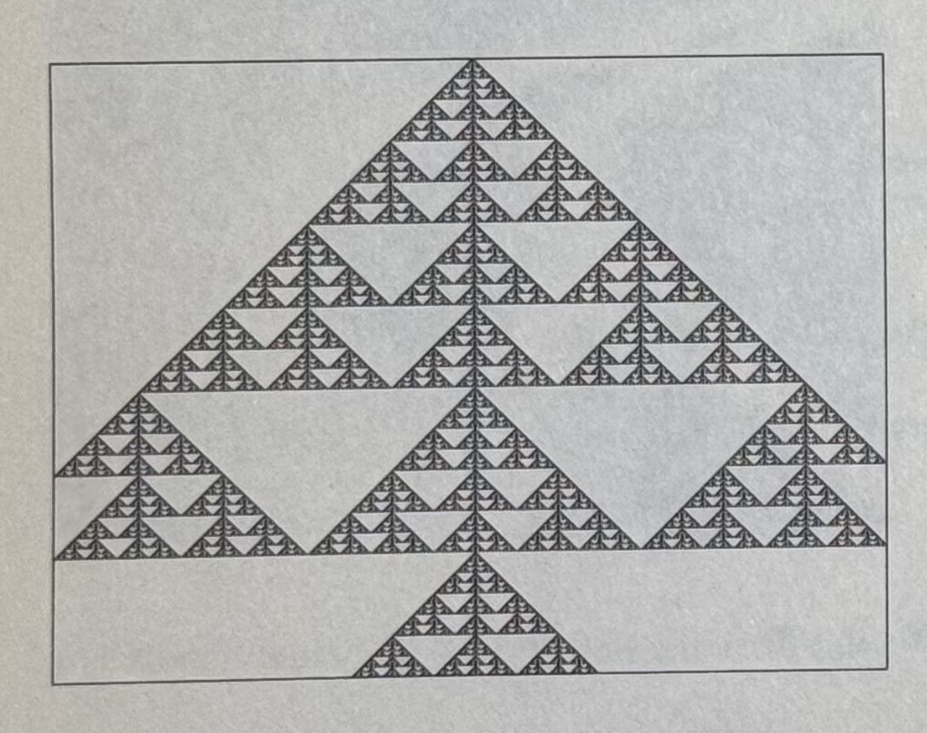


Bild 18.5 Zellularautomat von Wolfram

Die ersten Ansätze zur Automatentheorie stammen von John von Neumann (Ende der vierziger Jahre) und von A. Turing (1952). Sehr populär wurden die Zellularau-

tomaten durch das Spiel *Life* von John Conway und dessen Publikation durch Martin Gardner im Februarheft 1972 des *Scientific American*. Die grundlegende Arbeit über Zellularautomaten wurde von Stephan Wolfram in *Review of Modern Physics* (Juli 1983) publiziert; dieser Autor hat später auch das bekannte Programm *Mathematica* geschrieben.

Bei Zellularautomaten beschränken wir uns hier auf eindimensionale (lineare) und auf binäre; d.h. solche, die nur die Zustände 0 bzw. 1 annehmen können. Wolfram beschrieb u.a. folgenden Zellularautomaten an: *Eine Zelle wird genau dann lebendig (Zustand 1), wenn genau einer ihrer beiden Nachbarn lebt. Sie stirbt (Zustand 0), wenn beide Nachbarn leben oder tot sind.* Damit gibt es folgende Übergänge

$$00 \to 0; \quad 10 \to 1; \quad 01 \to 1; \quad 11 \to 0$$

Diese Zuordnung kann als Addition ⊕ modulo 2 oder als xor-Verknüpfung von Bits interpretiert werden. Die Produktionsregel des Automaten von Bild 18.5 ist somit

$$x_n \to x_{n-1} \oplus x_{n+1} = x_{n-1} \operatorname{xor} x_{n+1}$$

Aus dem Pascalschen Dreieck, bestehend aus den Binomialkoeffizienten, erhält man das Sierpinski-Dreieck, wenn man jeden Binomialkoeffizienten durch seinen Rest bei der Division durch 2 ersetzt. Die obige Formel generiert damit das Sierpinski-Dreieck. Einen ähnlichen Automaten erhält man, wenn man den vorhergehenden Zustand der Zelle selbst einbezieht

$$x_n \to x_{n+1} \oplus x_n \oplus x_{n+1}$$

Für drei Nachbarn gibt es $2^3 = 8$ Möglichkeiten.

$$111 \to a \quad 110 \to b \quad 101 \to c \quad 100 \to d$$
$$011 \to e \quad 010 \to f \quad 001 \to g \quad 000 \to h$$

Somit könnte man prinzipiell $2^8 = 256$ mögliche Regeln aufstellen. Nach Wolfram aber sind nur solche Regeln zugelassen, die weder links noch rechts bevorzugen; d.h. zu symmetrischen Ergebnissen führen. Damit bleiben nur noch 32 solcher Regeln übrig. Diese können, entsprechend der oben gemachten Aufzählung, beschrieben werden durch die Zahlenreihe

$$a \quad b \quad c \quad d \quad b \quad e \quad d \quad 0$$

Faßt man diese Zahlen als Stellen einer Binärzahl auf, so läßt sich jede Regel durch die so gewonnene Codierung C ausdrücken.

$$C = 128a + 64b + 32c + 16d + 8b + 4e + 2d$$

Dies soll am Beispiel des Sierpinski-Dreiecks erläutert werden. Gemäß den genannten Rechenregeln beim Pascalschen Dreieck erhält man

$$111 \to 0 \quad 110 \to 1 \quad 101 \to 0 \quad 100 \to 1$$
$$011 \to 1 \quad 010 \to 0 \quad 001 \to 1 \quad 000 \to 0$$

Hieraus ergibt sich die Zahlenfolge

$$0 \;\; 1 \;\; 0 \;\; 1 \;\; 1 \;\; 0 \;\; 1 \;\; 0 \cong 64+16+8+2=90$$

Dies bedeutend das Sierpinski-Dreieck erhält die Codierung 90. Entprechend liefert die Codierung 150 das Wolfram-Dreieck.

Das folgende Quick Basic-Programm `cell3.bas` startet jeweils mit einer Zelle und generiert so ein Dreieck z.B. nach Sierpinski oder Wolfram. Die gewünschte Produktionsregel ist als ganze Zahl ≤ 255 einzugeben, ebenso die gewünschte Zeilen- bzw. Spaltenzahl. Falls der Automat vorher abstirbt, wird die eingegebene Zeilenzahl nicht immer erreicht.

```
'cell3.bas
'Zellular-Automat mit 3 Nachbarn

CONST xm = 320: ym = 240: 'Bildschirm-Mitte
DIM i, j, m, n, p, q, s AS INTEGER

INPUT "Produktionsregel (ganze Zahl 0..255) eingeben"; c
INPUT "Wieviele Zeilen,Spalten"; p, n
n = n \ 2: p = p \ 2

DIM x(2 * n), y(2 * n), b(7)
SCREEN 12: CLS

m = c:'Binärentwicklung
FOR j = 0 TO 7
   q = INT(m / 2): b(j) = m - 2 * q: m = q
NEXT j
x(0) = 0: x(2 * n) = 0: x(n) = 1: PSET (xm, 0)
FOR j = 1 TO 2 * p + 1
   FOR i = 1 TO 2 * n - 1
         s = x(i + 1) + 2 * x(i) + 4 * x(i - 1): 'Produktions regel
         y(i) = b(s)
   NEXT i
   FOR i = 1 TO 2 * n - 1
      x(i) = y(i)
      IF x(i) = 1 THEN PSET (xm - n + i, j)
   NEXT i
NEXT j
e$ = INPUT$(1): SCREEN 0
END
```

18.6 Zellularautomaten mit 5 Nachbarn

Die Theorie von Wolfram kann direkt auf Zellularautomaten mit 5 Nachbarn verallgemeinert werden. Läßt man bei solchen Automaten für die Produktionsregeln nur Summenterme zu, so ist sein Schicksal eindeutig vorherbestimmt. Wolfram nennt solche Automaten daher »totalitär«. Eine Summe der Form

$$s = x_{n-2} + x_{n-1} + x_n + x_{n+1} + x_{n+2}$$

kann daher nur die Werte 0..5 annehmen. Codiert man diese Werte gemäß

$$5 \to a \quad 4 \to b \quad 3 \to c \quad 2 \to d \quad 1 \to e \quad 0 \to 0$$



Bild 18.6 Zellularautomat nach Regel 28

so gibt es $2^5 = 32$ Regeln, die wieder nach folgendem Schema codiert werden

$$C = 32a + 16b + 8c + 4d + 2e$$

Das Muster 101010 führt beispielsweise zur Codierung $C = 42$. Für die Codierung gibt es damit insgesamt 64 Werte; da $2e$ sicher gerade ist, kann eine Produktionsregel (wegen der Binärdarstellung) nur eine gerade Zahl ≤ 64 sein.

Das Turbo Pascal-Programm `cel15.pas` erzeugt Zellularautomaten gemäß den beschriebenen Regeln bei 5 Nachbarn. Damit der Bildschirm vollständig gefüllt wird, wird die erste Zellgeneration mit Zufallszahlen ausgelost. Der untere Rand des Bildschirms wird natürlich nur dann erreicht, wenn der Automat nicht vorher abstirbt.

```
program cell5;{ Zellularautomat mit 5 Nachbarn }
uses crt,graph;

const xm = 320; ym = 240; { Bildschirm-Mitte }
var c,i,j,m,q,s : integer;
    x,y : array[0..2*xm+2] of integer;
    b : array[0..5] of integer;
    Graphdriver,Graphmode:integer;

begin
write('Produktionsregel(gerade ganze Zahl 0..64) eingeben! ');
readln(c);
GraphDriver := Detect;
Initgraph(GraphDriver,GraphMode,'\tp\bgi');
SetgraphMode(Graphmode);
randomize;

m := c;
for j := 0 TO 5 do { Binärcodierung }
  begin
  q := m div 2; b[j] := m-2*q; m := q
  end;
x[0] := 0; x[1] := 0;
x[2*xm+1] := 0; x[2*xm+2] := 0;
for i := 2 to 2*xm do
  begin
  x[i] := random(2);
  if x[i]=1 then putpixel(i-1,0,13);
  end;
for j := 1 to 2*ym+1 do
begin
  for i := 2 to 2*xm do
    begin
    s := x[i-2] + x[i-1] + x[i] + x[i+1] + x[i+2]; { Produktionsregel }
    y[i] := b[s]
    end;
  for i := 2 to 2*xm do
    begin
    x[i] := y[i];
    if x[i]=1 then putpixel(i-1,j,13);
    end
end;
repeat until keypressed;
closegraph;
textmode(lastmode)
end.
```

Mit Hilfe der angegebenen Regeln gelang es Wolfram, die totalitären Zellularautomaten vollständig zu klassifizieren. Anschaulich gesprochen lassen sich diese Automaten in vier Klassen einteilen

Klasse	Struktur	Regelbeispiele
1	vollständig homogen	0,16,32,36,48
2	einfache periodische Strukturen	8,24,40,56,58
3	fraktale Strukturen	6,10,12,14,18
4	komplexe quasiperiodische Strukturen	20,52

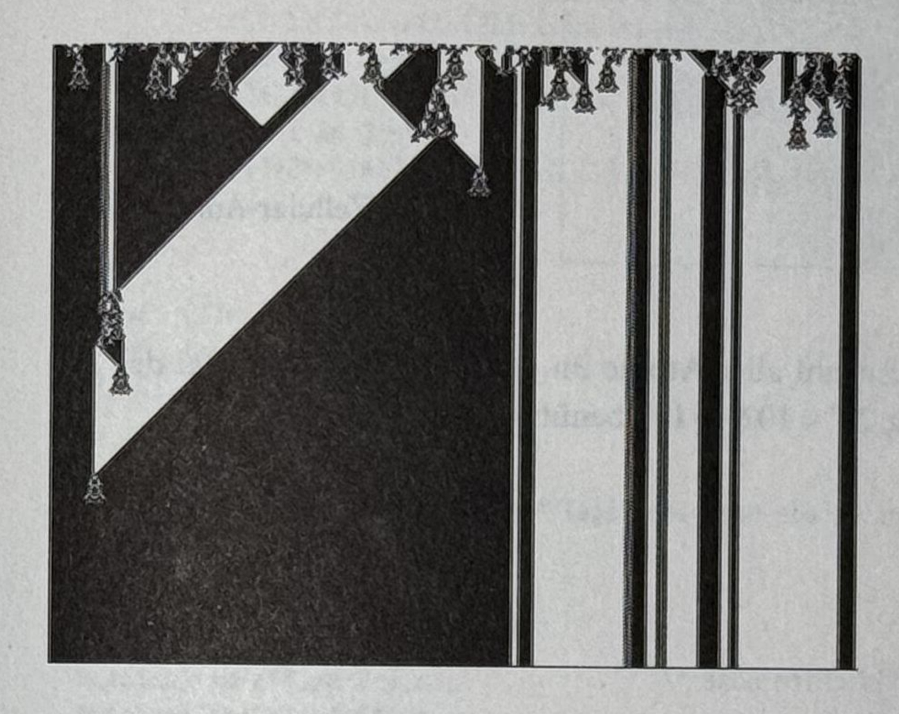


Bild 18.7 Zellularautomat nach Regel 52

18.7 Zellularautomat von Pickover

In der Zeitschrift *Algorithm* hat C. Pickover 1990 einen interessanten Algorithmus namens Gerade-ungerade-Wachstum *(even-odd-growth)* angegeben, der einen schönen Automaten erzeugt. Er beruht darauf, daß abwechselnd mit 4 Nachbarn (*von-Neumann*-Umgebung) und 8 Nachbarn (*Moore*-Umgebung) gerechnet wird. Die relativ komplizierte Produktionsregel kann direkt dem Listing des Turbo C-Programms `pickovr.c` entnommen werden.

Automaten mit Moore-Umgebungen können beliebig komplex werden. Dies kann man an einem kleinen Beispiel abschätzen. Hat ein Automat n Zustände und k Nachbarn, so gibt es (ohne einschränkende Symmetrieregeln) insgesamt n^{n^k} mögliche Regeln. Für einen binären Zellularautomaten mit Moore-Umgebung gibt es damit

$$2^{2^8} = 2^{256} = 2^{10 \cdot 25.6} \approx 10^{3 \cdot 25.6} \approx 10^{77}$$

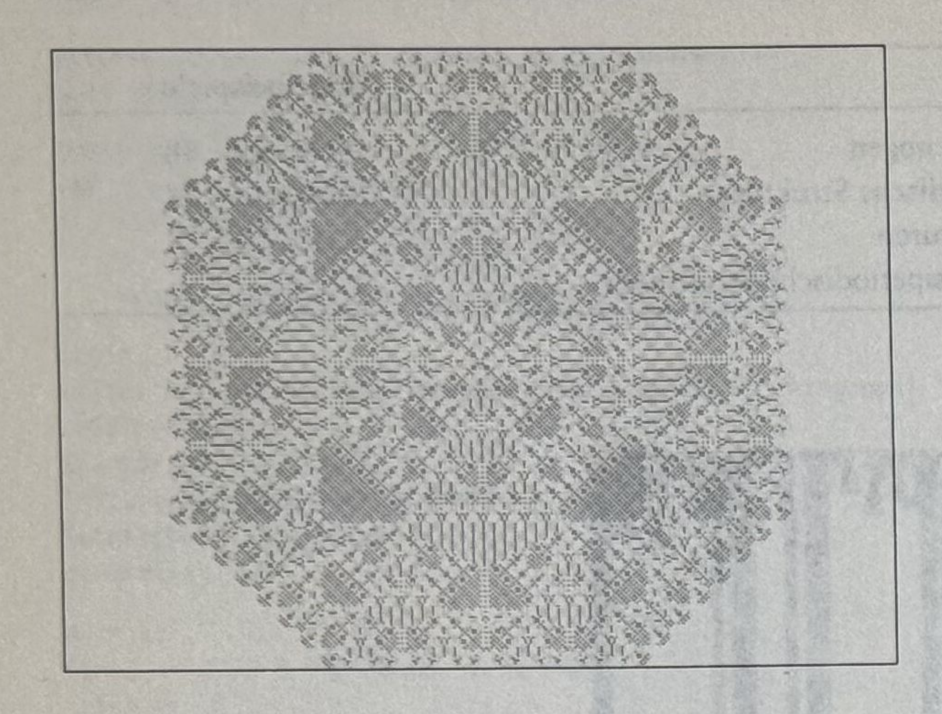


Bild 18.8 Zellular-Automat von Pickover

Zustände, dies entspricht der Anzahl aller Atome im ganzen Universum! Bei der Abschätzung wurde die Näherung $2^{10} = 1024 \approx 10^3$ benützt.

```
/* pickovr.c */
/* Zellular-Automat nach Pickover: gerade,ungerade-Regel */

#include <stdio.h>
#include <graphics.h>
#include <conio.h>
#define N 120   /* max.1/4 der Bildschirm-Höhe */
char x[N][N],y[N][N];
const int xm=320,ym=240; /* Bildschirm-Mitte */

void main(void)
{
int k,s=0;
register int i,j;
int gdriver,gmode;
gdriver = 9; gmode = VGAHI; /* VGA-Modus 640 x 480 */
initgraph(&gdriver,&gmode," ");
rectangle(0,0,639,479);

x[0][0] = 1;
for(k=1; k<N; k++)
{
for(i=0; i<=k; i++)
for(j=0; j<=k; j++)
if (x[i][j]==0)
  {
  if (k % 2 == 1)
    {
    if (i>=1 && j>=1)
      s = x[i+1][j]+x[i-1][j]+x[i][j+1]+x[i][j-1];
```

```
    else if (i==0 && j>=1)
       s = 2*x[1][j]+x[0][j+1]+x[0][j-1];
    else if (i>=1 && j==0)
       s = 2*x[i][1]+ x[i+1][0]+x[i-1][0];
    y[i][j]=s;
    }
    else
    {
    if (i>=1 && j>=1)
     s = x[i+1][j]+x[i-1][j]+x[i][j+1]+x[i][j-1]+x[i+1][j+1]+
            x[i-1][j+1]+x[i-1][j-1]+x[i+1][j-1];
    else if (i==0 && j>=1)
 s = 2*x[1][j]+2*x[1][j+1]+2*x[1][j-1] +x[0][j+1] +x[0][j-1];
    else if (i>=1 && j==0)
  s = 2*x[i][1]+2*x[i+1][1]+2*x[i-1][1]+x[i+1][0]+x[i-1][0];
    y[i][j]=s;
    }
  }
else y[i][j]=x[i][j];
for(i=0; i<=k; i++)
for(j=0; j<=k; j++)
{
int c;
static int f[] = {0,10,9,14,13,12,6,15};
if (k % 2==1)
  {
  c = f[y[i][j]];
  putpixel(xm+2*i,ym-2*j,c);
  putpixel(xm-2*i,ym-2*j,c);
  putpixel(xm+2*i,ym+2*j,c);
  putpixel(xm-2*i,ym+2*j,c);
  }
x[i][j] = (y[i][j]==1) ? 1 : 0;
}
}

do {} while(!(kbhit()));
closegraph();
return;
}
```

18.8 Cluster-Bildung (DLA)

Das Bilden von Clustern findet z.B. statt bei Anlagern von Teilchen an Oberflächen, elektrischen Entladungen, Wachstum von Dentriten und Diffusion von Wasser in Öl. Im Englischen spricht man hier von *Diffusion Limited Aggregation (DLA)*, etwa »durch Diffusion begrenztes Wachstum«.

Der zugrundeliegende Mechanismus der Clusterbildung ist theoretisch nicht in jedem Fall verstanden, daher ist eine Computer-Simulation von großem Interesse. Eine solche Simulation ist sehr rechenaufwendig, die Figuren in Bild 18.9 benötigen ca. 2 Stunden Rechenzeit auf einem Intel 80486-Rechner.

In allen Fällen muß mit einem geeigneten Verfahren geprüft werden, ob die entstehenden Cluster wirklich fraktale Eigenschaften haben. Dies kann mit dem im Abschnitt 5.4 angegebenen Box-Counting-Algorithmus erfolgen. Typische fraktale Cluster weisen etwa die Dimension des Sierpinski-Dreiecks mit $D = 1.58$ auf. Sehr dichte Schneeflocken haben eine fraktale Dimension von $D = 1.85 \pm 0.06$ (vgl. E.Stanley: *Fractals and Multifractals*, im Sammelband *Fractals and Disordered Systems*, Springer 1991 (Ed. A. Bunde/ S. Havlin)).

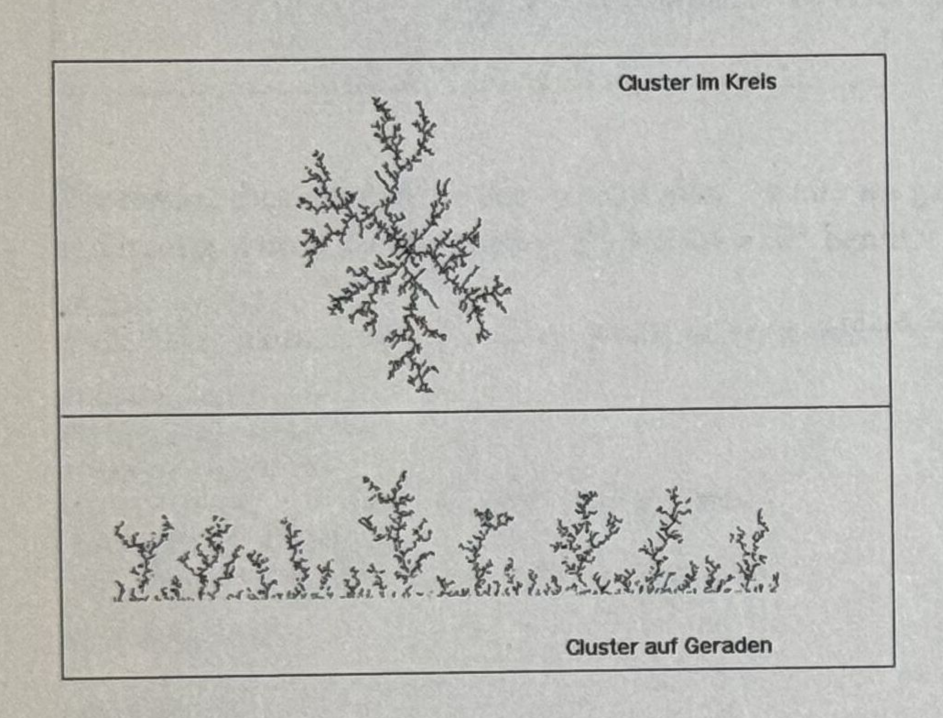


Bild 18.9 Cluster im Kreis und auf Geraden

Das folgende Quick Basic-Programm `clust1.bas` erlaubt die Simulation des radialen Cluster-Wachstums (in einem Kreis). Der Algorithmus ist relativ einfach. Zunächst wird ein beliebiger Punkt zufällig ausgelost und innerhalb seiner von-Neumann-Umgebung verschoben. Sodann wird geprüft, ob sich in seiner Moore-Umgebung ein Clusterteilchen befindet. Ist dies der Fall, so wird der Punkt gesetzt und der Abstand (zum Bildschirm-Mittelpunkt) aktualisiert, andernfalls wird der Punkt gelöscht; d.h. es wird ein neuer Punkt ausgelost. Die Entfernung der auszulosenden Punkte wird über den Parameter p gesteuert.

```
'clust.bas
'Clusterbildung im Kreis

CONST PI = 3.141529653#
CONST xm = 320: ym = 240: 'Bildschirm-Mitte
```

```
CONST p = 0.04: 'Abstandsparameter
DIM dir, dist, s, x, y AS INTEGER
DIM z AS LONG
DIM alpha, r AS SINGLE

SCREEN 12: CLS
RANDOMIZE TIMER
dist = 0: z = 0
PSET (xm, ym), 14

naechst:
z = z + 1
r = dist * (1 + RND *p): 'Zufallsabstand
alpha = 2 * PI * RND: 'Zufallswinkel im Bogenmaß
x = xm + INT(r * COS(alpha))
y = ym + INT(r * SIN(alpha)): 'Zufallspunkt (x,y)
DO WHILE INKEY$=""
  dir = INT(4 * RND): 'Zufallsrichtung
  SELECT CASE dir
    CASE 0: x = x - 1
    CASE 1: x = x + 1
    CASE 2: y = y - 1
    CASE 3: y = y + 1
  END SELECT
  'Untersuchen der Moore-Umgebung
  s = POINT(x - 1, y): s = s + POINT(x + 1, y)
  s = s + POINT(x, y - 1): s = s + POINT(x, y + 1)
  s = s + POINT(x + 1, y + 1): s = s + POINT(x - 1, y - 1)
  s = s + POINT(x - 1, y + 1): s = s + POINT(x + 1, y - 1)
  IF s > 0 THEN
    PSET (x, y), 1 + z \ 1000
    'Aktualisieren des Abstands
    IF dist < ABS(x - xm) THEN dist = ABS(x - xm):
    IF dist < ABS(y - ym) THEN dist = ABS(y - ym)
    GOTO naechst
   END IF
   IF (ABS(x - xm) > dist + 4) OR (ABS(y - ym) > dist + 4) THEN GOTO naechst
  LOOP
A$ = INPUT$(1): SCREEN 0
END
```

Ähnlich funktioniert das Verfahren für die Clusterbildung an einer geraden Kante. Der von der Prozedur `naechstpunkt` ausgeloste Punkt wird zufällig verschoben (in seiner von-Neumann-Umgebung). Die Funktion `nachbarn` übernimmt dann die Zählung der (etwaigen) Nachbarn. Findet sich ein Nachbarpunkt, so wird der Punkt angelagert; andernfalls der nächste Punkt gewählt. Das Verfahren kann vom Turbo Pascal-Program `clust2.pas` durchgeführt werden.

```
program clust2; { Clusterbildung auf Geraden }
uses crt,graph;

label naechst;
const xm = 320; ym = 400; { Startzeile ym }
var dist,x,y : integer;
    z : longint;
    Graphdriver,Graphmode:integer;

function nachbarn(x,y:integer):integer;
{ Untersuchen der Moore-Umgebung}
var s: integer;
begin
s := getpixel(x-1,y);
s := s + getpixel(x+1,y);
s := s + getpixel(x,y-1);
s := s + getpixel(x,y+1);
s := s + getpixel(x+1,y+1);
s := s + getpixel(x-1,y-1);
s := s + getpixel(x-1,y+1);
nachbarn := s + getpixel(x+1,y-1);
end;

procedure naechstpunkt(var x,y:integer);
var dir : integer;
begin
dir := random(4); { Zufallsrichtung }
case dir of
0: begin
   x := x-1; if x<40 then x := x + 2;
   end;
1: begin
   x := x+1;
   if x > 600 then x := x - 2;
   end;
2: begin
   y := y-1;
   if y < ym-dist-12 then y := y + 2;
   end;
3: y := y+1 end;
end;

begin
GraphDriver := Detect;
Initgraph(GraphDriver,GraphMode,'\tp\bgi');
SetgraphMode(Graphmode);
rectangle(0,0,639,479);
randomize;
dist := 0; z := 0;
putpixel(xm,ym,14);

naechst:
z := z+1;
```

```
x := xm+trunc((random-0.5)*560);
y := ym-dist-6;  { Zufallspunkt (x,y)}
repeat
   naechstpunkt(x,y);
   if nachbarn(x,y)>0 then
       begin
       putpixel(x,y,14);
       if dist<ym-y then dist := ym-y;
       goto naechst;
       end;
   if y = ym then
   begin putpixel(x,y,14); goto naechst end;
until keypressed;
repeat until keypressed;
closegraph;
textmode(lastmode)
end.
```

19 Kleines Lexikon der Chaostheorie

Attraktor: Ein Attraktor eines dynamischen Systems $\mathbf{f}^t$ ist nach J. P. Eckmann eine kompakte Menge A mit folgenden Eigenschaften

1. A ist invariant unter $\mathbf{f}^t$: $\mathbf{f}^t(A) = A$
2. A hat eine offene Umgebung, die sich unter $\mathbf{f}^t$ auf A zusammenzieht:
   $$\lim_{t \to \infty} \mathbf{f}^t(A) = A$$
3. A hat keine Untermenge, die transient ist. A kann nicht in nichttriviale, kompakte und invariante Mengen zerlegt werden; d.h. kann nicht in separate Attraktoren zerlegt werden.

Autonomes System: Ein dynamisches System heißt autonom, wenn es nicht explizit von der Zeit abhängt. Autonome Systeme haben gegenüber heterogenen folgende Vorteile:

1. Die Trajektorien sind eindeutig festgelegt durch die Anfangsbedingungen
2. Die Trajektorien können sich nicht schneiden.

Attraktionsgebiet: Das Attraktorgebiet oder Bassin eines Attraktors ist die offene Menge aller Anfangsbedingungen $\mathbf{x}_0$, für die gilt: $\lim_{t \to \infty} \mathbf{f}^t(\mathbf{x}_0) \in A$.

Bahn: Die Menge der Punkte $\left\{\mathbf{f}^t\right\}_{t=0}^{t=n}$ mit $\mathbf{f}(t_0) = \mathbf{x}_0$ heißt die Bahn, *Orbit* oder *Trajektorie* zur Anfangsbedingung $\mathbf{x}_0$.

Box-Count-Dimension: Wird eine Menge von N_ε Quadraten bzw. Würfeln der Kantenlänge ε überdeckt, so heißt der Grenzwert

$$D = \lim_{\varepsilon \to \infty} \frac{\ln N_\varepsilon}{\ln \frac{1}{\varepsilon}}$$

die Box-Count-Dimension oder Kapazität. Sie ist ein Spezialfall der Hausdorff-Besikowitch-Dimension.

Chaos-System: Folgende Eigenschaften eines dynamischen Systems werden i.a. als kennzeichnend angesehen:

1. Der Bereich des Phasenraums, der von den Trajektorien angelaufen wird, ist beschränkt.
2. Die Trajektorien sind nicht periodisch.
3. Ein Attraktor des System hat i.a. fraktale Dimension.
4. Das System hängt empfindlich von den Anfangsbedingungen ab. D.h. die Trajektorien zweier beliebig benachbarter Anfangszustände divergieren nach endlicher Zeit. Die Divergenzrate kann mithilfe des Ljapunow-Exponenten gemessen werden.

Correlationsintegral: Das Correlationsintegral einer Punktmenge ist definiert durch

$$C(r) = \lim_{n\to\infty} \frac{1}{n^2}\sum_i , \sum_{j=1;\, i\neq j}^{n} H\left(r - \left|x_i - x_j\right|\right)$$

Dabei ist die Heavyside-Funktion erklärt als

$$H(x) = \begin{cases} 0 & \textit{für } x \leq 0 \\ 1 & \textit{für } x > 0 \end{cases}$$

Correlationsdimension: Stellt man das Correlationsintegral als Potenzgesetz dar $C(r) \propto r^D$, so heißt der Exponent D die Correlationsdimension. Dieses Verhalten läßt sich auch als Grenzwert schreiben

$$D = \lim_{r\to 0} \frac{\ln C(r)}{\ln r}$$

Dimensionsbegriff: Jeder Dimensionsbegriff sollte über folgende Eigenschaften verfügen:

1. $E \subset F \Rightarrow \dim(E) \leq \dim(F)$ *Montonie*
2. $\dim(E \cup F) = \max(\dim(E), \dim(F))$ *Stabilität*
3. $\dim\left(\bigcup_{i=1}^{n} F_i\right) = \sup_{1\leq i<\infty} \dim(F_i)$ *Stabilität bei abzählbaren Mengen*
4. Ist f eine Ähnlichkeitsabbildung, so gilt $\dim f(E) = \dim(E)$ *Invarianz*

5. $E \subset R^n \Rightarrow \dim(E) = n$ *offene Teilmenge des* R^n
6. Für glatte Mannigfaltigkeiten wie Geraden, Ebenen, Würfel stimmt die Dimension mit der Euklidischen überein.

Dissipatives System: Ein System heißt dissipativ, wenn sich ein Volumenelement des Phasenraums auf einen Punkt zusammenzieht. Es gilt dann $div\,\mathbf{F} < 0$. Dies ist insbesondere bei physikalischen Systemen mit Reibung der Fall. Die Volumenkontraktion wird durch einen negativen Ljapunow-Exponenten beschrieben. Vergleiche dazu konservative Systeme.

Durchmesser: Ist U eine nichtleere Teilmenge des R^n, so heißt

$$|U| = \sup\{|x-y|; x, y \in U\}$$

der Durchmesser von U.

Dynamisches System: Ein dynamisches System ist bestimmt durch ein System von Differenzengleichungen

$$\mathbf{x}(t+1) = \mathbf{f}(\mathbf{x}, t) = \mathbf{f}^t(\mathbf{x})$$

oder Differentialgleichungen (auch *Fluß* genannt)

$$\dot{\mathbf{x}} = \mathbf{F}(\mathbf{x})$$

Elliptisch: Ein Fixpunkt heißt elliptisch, wenn die Realteile der zugehörigen Eigenwerte verschwinden.

Entropie: Der Ausdruck

$$E = \sum_{i=1}^{N} \frac{1}{p_i} \ln p_i$$

kann wahlweise als Entropie oder als Informationsgehalt eines Systems interpretiert werden, je nachdem wie die Wahrscheinlichkeit p_i gedeutet wird.

Feigenbaum-Bifurkation: Ein System habe bis zu einem Parameterwert r einen stabilen Fixpunkt. Wird dieser Fixpunkt bei einem weiteren Anwachsen des Parameterwertes instabil und zerfällt in zwei stabile Fixpunkte, so spricht man von einer Feigenbaum- oder *Gabel-Bifurkation.*

Feigenbaum-Konstante: Ist r_n die n-te Bifurkationsstelle eines dynamischen Systems, so existiert für eine Klasse von Funktionen der gemeinsame Grenzwert

$$F = \lim_{n\to\infty} \frac{r_n - r_{n-1}}{r_{n+1} - r_n} = 4.669202..$$

Dieser Grenzwert wird Feigenbaum-Konstante genannt.

Fixpunkt: Ein Punkt $\mathbf{x} \in R^n$ heißt Fixpunkt, wenn er die Bedingung $\mathbf{x} = \mathbf{F}(\mathbf{x})$ erfüllt.

Fraktal: Eine Punktemenge $F \in R^n$ heißt ein Fraktal nach K.Falconer[07], wenn gilt

1. F hat eine Feinstruktur; d.h. sie zeigt auf beliebig kleinen Skalen noch Struktur.
2. F ist zu irregulär, um lokal oder global mit der Euklidischen Geometrie beschrieben werden zu können.
3. F zeigt eine exakte oder angenäherte Selbstähnlichkeit.
4. F hat eine fraktale Dimension, die meist die Eukidische übersteigt. F kann auf einfache Weise definiert werden, meist rekursiv.

Fraktale Dimension: Zerfällt eine selbstähnliche Menge bei einer zentrischen Streckung mit dem Streckfaktor $k = \frac{1}{r}$ in N gleichartige Teile, so ist die fraktale Dimension

$$D = \frac{\ln N}{\ln \frac{1}{r}}$$

Geschlossene Bahn: Eine Trajektorie $\mathbf{x}(t)$ eines dynamischen Systems heißt geschlossene Bahn oder invarianter Kreis, wenn für ein t gilt

$$\mathbf{f}^t(\mathbf{x}) = \mathbf{x}$$

und $\mathbf{x}(t)$ kein Gleichgewichts- oder Fixpunkt ist.

Grenzsystem: Das Grenzsystem eines 3-dimensionalen dynamischen Systems kann sein:

1. Fixpunkt (alle Ljapunow-Exponenten <0)
2. Grenzzyklus (ein Ljapunow-Exponent =0)

3. Torus (zwei Ljapunow-Exponenten =0)
4. Attraktor (Ljapunow-Exponenten sind <0,=0 bzw.>0)

Grenzzyklus: Eine geschlossene Bahn $\mathbf{x}(t)$ heißt ein Grenzzyklus, wenn alle Lösungen des dynamischen Systems, die durch die Punkte einer Umgebung verlaufen, gegen $\mathbf{x}(t)$ konvergieren.

Hausdorff-Besikowitch-Dimension: Sei E eine Teilmenge des R^n und $\{U_i\}$ eine ε-Überdeckung von E. Für eine nichtnegative Zahl r sei das Infimum der Durchmesser definiert

$$H_\varepsilon^r(E) = \inf \sum_{i=1}^{n} \left|U_i\right|^r$$

Dann heißt die Zahl

$$D = \inf\left\{s: H_\varepsilon^s(E) = 0\right\} = \sup\left\{s: H_\varepsilon^s(E) = \infty\right\}$$

die Hausdorff-Besikowitch-Dimension.

Heteroklin: Die Bahn in einem dynamischen System heißt heteroklin, wenn sie an verschiedenen Fixpunkten beginnt und endet.

Heteronomes System: Ein dynamisches System $\mathbf{f}(\mathbf{x},t)$ heißt heteronom, wenn es explizit von der Zeit abhängt. Jedes heteronome System kann durch die Hinzunahme einer weiteren Variablen autonom gemacht werden. Ist $\mathbf{f}(\mathbf{x},t)$ periodisch mit der Periode T, so führt man die Variable $\theta = 2\pi \frac{t}{T}$ ein. Das so entstehende autonome System ist 2π-periodisch und hat den Phasenraum $R^n \times S$, wobei $S = [0,2\pi)$ der Einheitskreis ist.

Homoklin: Die Bahn in einem dynamischen System heißt homoklin, wenn sie an einem bestimmten Fixpunkt beginnt und endet.

Hopf-Bifurkation: Benannt nach dem deutschen Mathematiker Eberhard Hopf (1948). Voraussetzung sei, daß alle Eigenwerte eines Systems negative Realteile haben mit Ausnahme eines einzigen Paares konjugiert komplexer Eigenwerte $\lambda = a(r) + ib(r)$; $\bar{\lambda} = a(r) - ib(r)$. Überschreiten diese Eigenwerte für einen Parame-

terwert r die imaginäre Grenze, d.h. es wird $a(r) \geq 0$, so zerfällt ein stabiler Fixpunkt in einen stabilen Grenzzyklus.

Hyperbolisch: Ein Fixpunkt heißt hyperbolisch, wenn die Realteile der zugehörigen Eigenwerte nicht verschwinden.

Invarianter Kreis: Siehe geschlossene Bahn.

KAM-Theorem: Benannt nach Kolmogorow (1954), Arnold (1963) und Moser (1973). Als bestimmend für die Konvergenz der Reihenentwicklung eines gestörten Hamilton-Systems hatte Birkhoff den Term

$$\varepsilon \sum \frac{A\left[\omega_1, \omega_2, \cdots, \omega_f\right]}{m_1\omega_1 + m_2\omega_2 + \cdots + m_f\omega_f}$$

angegeben. Verschwindet die Summe im Nenner $\sum m_i\omega_i = 0$ (*Resonanz-Bedingung*) für einen Satz von ganzen Zahlen m_i, so tritt keine Konvergenz ein. Das KAM-Theorem liefert nun eine genauere Abschätzung für die Konvergenz. Gilt

$$\left|\sum_{i=1}^{f} m_i\omega_i\right| > \beta(\varepsilon)\left[\sum |m_i|\right]^{-1-\delta}$$

für alle ganze Zahlen m_i,so kann das gestörte Hamilton-System integriert werden. Für Systeme mit $f = 2$ Freiheitsgrade reduziert sich diese Bedingung auf

$$\left|\frac{\omega_1}{\omega_2} - \frac{m_1}{m_2}\right| > \beta(\varepsilon)$$

Im Falle des Anwachsens der Störung werden zuerst die Bewegungen chaotisch, für die das Frequenzverhältnis $\omega_1 : \omega_2$ nahezu rational ist.

Kapazität: Siehe Box-Count-Dimension.

Kaplan-Yorke-Dimension: Ordnet man die Ljapunow-Exponenten eines Systems monoton fallend, so gilt für die Kaplan-Yorke-Dimension (auch Ljapunow-Dimension genannt)

$$D = j + \frac{l_1 + l_2 + \cdots + l_j}{\left|l_{j+1}\right|}$$

dabei ist der Index j gegeben durch die Ungleichungen

$$\sum_{i=1}^{j}\lambda_i > 0 \wedge \sum_{i=1}^{j+1}\lambda_i < 0$$

Konservatives System: Ein System **F** heißt konservativ, wenn die Divergenz des zugehörigen Vektorfeldes verschwindet:

$$div\,\mathbf{F} = \sum_{i=1}^{n}\frac{\partial F_i}{\partial x_i} = 0$$

Dies ist insbesondere bei physikalischen Systemen ohne Reibung der Fall. Das Volumen eines Volumenelements im Phasenraums bleibt konstant. Vergleiche mit dissipativen Systemen.

Ljapunow-Exponent: Ein Ljapunow-Exponent λ eines dynamischen Systems kennzeichnet das Verhalten des Phasenraums. $\lambda < 0$ zeigt die Kontraktion, $\lambda > 0$ die Expansion. Für endliche Grenzsysteme (siehe dort) muß daher gelten

$$\sum \lambda_i < 0$$

d. h.die Kontraktion überwiegt.

Periodenverdopplung: Unter Periodenverdopplung versteht man den Zerfall eines stabilen Fixpunktes in zwei stabile Fixpunkte. Siehe auch Feigenbaum-Bifurkation.

Periodisches System: Ein System heißt periodisch mit der Periode T, wenn gilt: $\mathbf{f}(\mathbf{x}_0, t) = \mathbf{f}(\mathbf{x}_0, t+T)$

Phasenraum: Hat ein System von n Freiheitsgraden die Zustandsvariablen $x_i(t)$, so heißt der von den Variablen

$$\left\{x_i(t), \frac{\partial x_i}{\partial t}\right\}_{i=1}^{n}$$

aufgespannte $2n$-dimensionale Zustandsraum der Phasenraum des Systems.

Poincaré-Abbildung: Betrachtet werde ein heterogenes, periodisches dynamisches System der Dimension n. Dieses kann auf ein autonomes System erweitert werden, dessen Trajektorien auf dem $(n+1)$-dimensionalen Zylinder $R^n \times S$ liegen. Ein Schnitt durch diesen Zylinder bei festem $\theta = \theta_0$ erzeugt die Hyperebene

$$\Sigma = \left\{ (\mathbf{x}, \theta) \in R^n \times S; \theta = \theta_0 \right\}$$

Die Trajektorien schneiden Σ für $\theta = \theta_0 + 2\pi$. Damit ist die Poincaré- Abbildung P der Hyperebene Σ auf sich selbst definiert mittels

$$P(\mathbf{x}) = \mathbf{f}(\mathbf{x}, t_0)$$

Die Poincaré-Abbildung P ist topologisch äquivalent zum Trajektorienverlauf im Phasenraum. Dies bedeutet

1. Für konservative Systeme erhält P die Flächeninhalte in Σ
2. Für eine geschlossene Bahn enthält P genau einen Punkt
3. Die Poincaré-Abbildung eines p-periodischen Systems besteht genau aus p Punkten.
4. Systeme mit quasi-periodischen Lösungen haben Trajektorien auf einem zweidimensionalen Torus $S \times S$. Der Poincaré-Schnitt liefert hier einen Kreis bzw. Halbkreis.
5. Hat das ursprünglich autonome System einen Attraktor, so findet sich dessen Struktur auch im Poincaré-Schnitt (vgl. Bild 19.1).

Poincaré-Schnitt: Die Menge aller Punkte, die durch Hintereinanderausführen der Poincaré-Abbildung

$$\left\{ P^k(\mathbf{x}) \right\}_{k=1}^{\infty}$$

erzeugt werden, heißt Poincaré-Schnitt.

Quasiperiodische Lösung: Siehe Torus

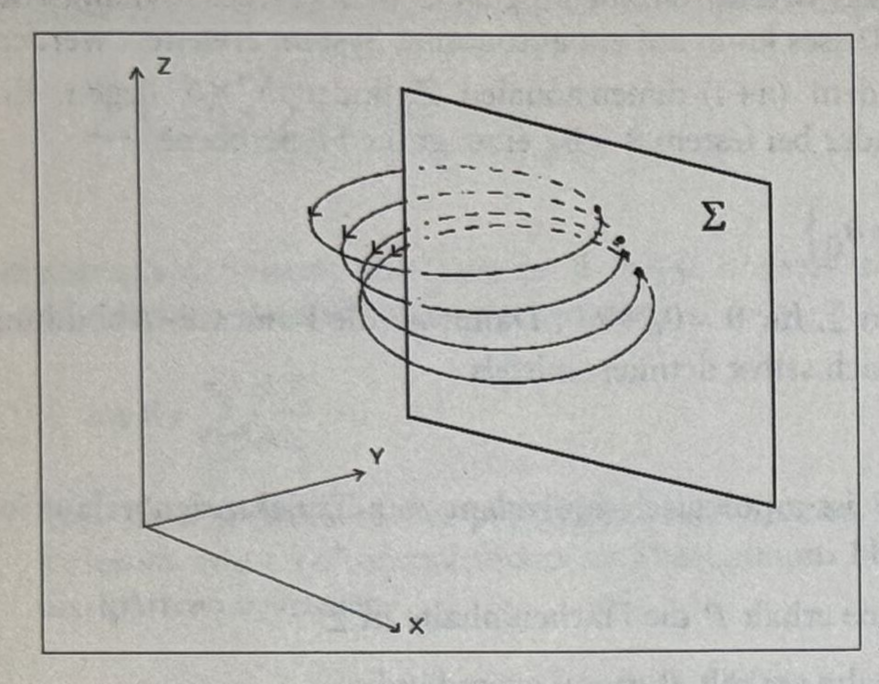


Bild 19.1 Poincaré-Abbildung

Sarkowskii-Theorem: Die Sarkowskii-Anordnung der natürlichen Zahlen lautet

$$3 \triangleright 5 \triangleright 7 \triangleright 9 \triangleright 11 \triangleright \ldots$$

$$2\cdot 3 \triangleright 2\cdot 5 \triangleright 2\cdot 7 \triangleright 2\cdot 9 \triangleright \ldots$$

$$2^2\cdot 3 \triangleright 2^2\cdot 5 \triangleright 2^2\cdot 7 \triangleright \ldots$$

$$2^3\cdot 3 \triangleright 2^3\cdot 5 \triangleright 2^3\cdot 7 \triangleright \ldots$$

....

$$\ldots \triangleright 2^n \triangleright \ldots \triangleright 2^3 \triangleright 2^2 \triangleright 2 \triangleright 1.$$

Hat eine eindimensionale,stetige Funktion die Periode p, so besitzt sie ebenfalls alle Perioden der Länge q, wobei $p \triangleright q$ gilt. Insbesondere zieht die Periode 3 alle anderen Perioden und damit Chaos nach sich.

Sattelpunkt-Bifurkation: Fällt bei einem bestimmten Parameterwert ein stabiler mit einem instabilen Fixpunkt zusammen, so spricht man von einer Sattelpunkt-Bifurkation.

Sensitive Abhängigkeit: Ein dynamisches System heißt sensitiv abhängig von den Anfangsbedingungen, wenn ein $\delta > 0$ so existiert, daß für ein beliebig benachbartes Element $\mathbf{y} \in U_\varepsilon(\mathbf{x})$ folgt

$$\left|\mathbf{f}^t(\mathbf{x}) - \mathbf{f}^t(\mathbf{y})\right| > \delta$$

Seltsamer Attraktor: Ein nichtperiodischer Attraktor heißt seltsam (englisch *strange*), wenn er sensitiv von den Anfangsbedingungen abhängt.

Stabiler Fixpunkt: Ein Fixpunkt $\mathbf{y}^*$ heiß stabil im Sinne von Ljapunow, wenn es für beliebige $\varepsilon > 0$ ein $\delta > 0$ so gibt, daß für die Lösungen $\mathbf{y}_0 = \mathbf{f}(\mathbf{x}, t_0)$ gilt

$$\left|\mathbf{f}(\mathbf{x}, t) - y^*\right| < \varepsilon \ \forall t > 0, \text{ falls gilt } \left|\mathbf{y}_0 - \mathbf{y}^*\right| < \delta .$$

Torus: Für ein System der Periode 2 liegen mit der Periode auch zwei Frequenzen Ω_1, Ω_2 fest.

1. Falls $\Omega_1 : \Omega_2 = n_1 : n_2$ rational ist, kann jede Trajektorie als Schraubenlinie auf einem Torus $\subset R^3$ interpretiert werden, die sich nach n_2 Umläufen und n_1 Windungen schließt.
2. Falls das Verhältnis $\Omega_1 : \Omega_2$ irrational ist, verlaufen die Trajektorien ebenfalls auf einem 2-dimensionalen Torus $S \times S$. Jedoch schließen sich die Bahnen nicht, sodaß die gesamte Torus-Oberfläche erfaßt wird. Diese Bewegung wird als quasi-periodisch bezeichnet.

Trajektorie: Siehe Bahn

Transiente Bahn: Die Bahn in einem Gebiet $G \subset R^n$ mit dem Anfangspunkt $\mathbf{x}_0 = f(\mathbf{x}, t_0)$ heißt transient, wenn gilt $x_0 \in G \backslash A$, wobei A ein Attraktor ist.

20 Literaturverzeichnis

[01] Baker G./ Gollub J.: *Chaotic Dynamics*, Cambridge University Press 1990

[02] Barnsley M.: *Fractals Everywhere*, Academic Press Boston 1988

[03] Briggs J./ Peat F.: *Die Entdeckung des Chaos*, Hanser 1990, dtv 1993

[04] Crilly A./ Earnshaw R./ Jones H. (Ed.): *Fractals and Chaos*, Springer New York 1991

[05] Cvitanovic P. [Ed.) : *Universality in Chaos*, Adam Hilger Bristol 1992

[06] Devaney R. L.: *A First Course in Chaotic Dynamical Systems*, Addison Wesley Reading 1992

[07] Falconer K.: *Fractal Geometry*, John Wiley Chicester 1990

[08] Field M./ Golubitzky M.: *Symmetry in Chaos*, Oxford University Press 1992

[09] Frφyland J.: *Introduction to Chaos and Coherence*, Institute of Physics Publishing Bristol 1992

[10] Gerok W. (Ed.): *Ordnung und Chaos*, Wissenschaftl. Verlagsgesellschaft Stuttgart 1989

[11] Gleick J.: *Chaos – die Ordnung des Universums*, Droemer-Knaur München 1990

[12] Heinrichs G.: *Chaos*, Aulis Köln 1993

[13] Holden A. (Ed.): *Chaos*, Manchester University Press 1987

[14] Lauwerier H.: *Fractals-Images of Chaos*, Penguin 1987

[15] Lauwerier H.: *Fraktale II*, Wittig Hückelhoven 1992

[16] Loistl O./ Betz I.: *Chaostheorie*, Oldenbourg München 1993

[17] Mandelbrot B.: *Die fraktale Geometrie der Natur*, Birkhäuser Basel 1987

[18] Mahnke R./ Schmelzer J./ Röpke G.: *Nichtlineare Phänomene und Selbstorganisation* Teubner Stuttgart 1992

[18] Mullin T.(Ed): *The Nature of Chaos*, Oxford Press Oxford 1993

[19] Ott E.: *Chaos in Dynamical Systems*, Cambridge University Press 1993

[20] Peitgen H./ Jürgens H./ Saupe D.: *Fractals for the Classroom I*, Springer New York 1992

[21] Peitgen H./ Jürgens H./ Saupe D.: *Fractals for the Classroom II*, Springer New York 1992

[22] Peitgen H./ Richter P.: *The Beauty of Fractals*, Springer New York 1986

[23] Peitgen H./ Saupe D.(Ed.).: *The Science of Fractal Images*, Springer New York 1988

[24] Pietronero L./ Tosatti E. (Ed.): *Fractals in Physics*, Elsevier Science Publishers 1986

[26] Prusinkiewicz P./ Hassan J.: *Lindenmayer Systems, Fractals und Plants*, Lecture Notes in Biomathematics 79 Springer New York 1989

[27] Prusinkiewicz P./ Lindenmayer A.: *The Algorithmic Beauty of Plants*, Springer New York 1990

[28] Rietman E.: *Exploring the Geometry of Nature*, Windcrest Blue Ridge Summit 1989

[29] Ruelle D.: *Chaotic Evolution and Strange Attractors*, Cambridge University Press 1987

[30] Schmitter E.: *Fraktale Geometrie*, Hofacker Holzkirchen 1989

[31] Schroeder M.: *Fractals, Chaos, Power Laws*, Freemann New York 1991

[32] Schuster H.: *Deterministic Chaos*, VCH Weinheim 1988

[33] Steeb W.: *Chaos and Fractals*, BI Wissenschaftsverlag Mannheim 1992

[34] Steeb W.: *A Handbook of Terms used in Chaos and Quantum Chaos*, BI Wissenschaftsverlag Mannheim 1991

[35] Steeb W./ Kunik A.: *Chaos in Dynamischen Systemen*, BI Wissenschaftsverlag Mannheim 1992

[36] Steward I.: *Does God play Dice? – The Mathematics of Chaos*, Penguin 1990

[37] Tufillaro N./ Abbott T./ Reilly J.: *An Experimental Approach to Nonlinear Dynamics and Chaos*, Addison Wesley Redwood City 1992

[38] Wiggins S.: *Introduction to Applied Nonlinear Dynamical System and Chaos*, Springer New York 1990

[39] Worg R.: *Deterministisches Chaos*, BI Wissenschaftsverlag Mannheim 1993

[40] Zeitler H./ Neidhardt W.: *Fraktale und Chaos*, Wissenschaftl. Buchgesellschaft Darmstadt 1993

Index

Lehrbücher



Betriebssysteme

Konzepte, Methoden und Modelle

Uwe Baumgarten/Peter Paul Spies

Das Buch zeigt die Linien der Entwicklungsgeschichte von Betriebssystemen auf. Es behandelt Modelle, Konzepte und Verfahren, die bei Entwürfen und Realisierungen von Betriebssystemen zur Lösung der vielfältigen Aufgaben anzuwenden sind. Die wesentlichen Eigenschaften werden ausführlich erklärt sowie an Beispielen veranschaulicht.

ca. 600 Seiten,1993,
ca. 59,90 DM, geb., ISBN 3-89319-318-9

Objektorientierte Entwicklung von Software-Systemen

Dr. Horst A. Neumann

ca. 450 Seiten, 1993
ca. 79,90 DM, ISBN 3-89319-452-5

Neuronale Netze

H. Ritter/Th. Martinetz/K. Schulten

330 Seiten, 2. überarb. Auflage 1991
49,00 DM, gebunden, ISBN 3-89319-131-3



EDV-Grundwissen

Eine Einführung in Theorie und Praxis der EDV

M. Precht/N. Meier/J. Kleinlein

Das Buch fundiert die theoretischen und technischen Grundlagen der Datenverarbeitung. Theorie und praktische Anwendung stehen gleichermaßen im Mittelpunkt

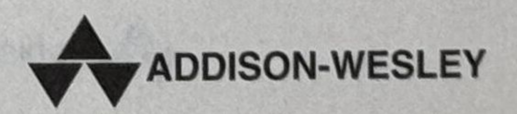
280 Seiten, 1992
49,90 DM, gebunden
ISBN 3-89319-413-4

Informationstheorie

Grundlage der (Tele-) Kommunikation

Rolf Johannesson

298 Seiten, 1992
69,90 DM, gebunden, ISBN 3-89319-465-7



Algorithmen für Chaos und Fraktale

Dietmar Herrmann

Die Chaostheorie und die fraktale Geometrie haben inzwischen ihren Siegeszug um die Welt angetreten. Kaum eine Zeitschrift versäumte es, Ihre Leser mit wunderbaren und kunstvollen Bildern an der Entdekkungsreise in das Reich des Apfelmännchens und der Julia-Mengen teilhaben zu lassen.

Zwischen den Anforderungen, die eine populäre, auf Anschauung gerichtete Einführung und eine rein wissenschaftlich orientierte Abhandlung stellt, klafft eine breite Lücke. Hier setzt das vorliegende Buch ein. Freunde der Computergrafik finden eine Vielzahl von Grafiken, Studierende der Naturwissenschaften zahlreiche Anwendungen und Dozenten Grundlagen für einen Kurs oder eine Vorlesung.

[illegible] Grundlagen [illegible] Theorie [illegible] einem mittleren Niveau und setzt dabei nur elementare Kenntnisse der Differential- [illegible]

Der Leser erhält fundierte Einblicke in Bereiche der dynamischen Systemtheorie wie Fixpunkt-Theorie, Ljapunow-Exponenten, Attraktoren oder KAM-Theorie. Das Buch ist eine wahre Fundgrube für mathematische Algorithmen aus den Bereichen Iterierte Funktionensysteme, Lindenmayer Systeme, Julia-Mengen, Mandelbrot-Mengen, Fraktale Kurven und Zellular-Automaten, die als Computerprogramme in den gängigen Programmiersprachen realisiert sind.

Besonderen Wert wurde auf die Darstellung von Anwendungen gelegt. Der Leser erfährt wichtige Auswirkungen des Chaos in der Natur aus den Bereichen wie Chemie, Physik, Elektronik, Astronomie usw. Ein Glossar über die wichtigsten Begriffe der dynamischen Systemtheorie schließt das Buch ab. Neben der Vielzahl der im Buch aufgelisteten Programme enthält die beigefügte Diskette zahlreiche weitere Programme, die aus Umfangsgründen [illegible] abgedruckt wurden.

Algorith[illegible]
Compute[illegible]
ISBN 3-89319-633-1 öS 467,00
VVA-Nr. 563-00633-5 sFr 57,90



